Proof Search in Propositional Circumscription and Default Logic

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Peppo Brambilla
von Bern

Leiter der Arbeit:
Prof. Dr. G. Jäger
Institut für Informatik und angewandte Mathematik

Proof Search in Propositional Circumscription and Default Logic

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Peppo Brambilla
von Bern

Leiter der Arbeit:
Prof. Dr. G. Jäger
Institut für Informatik und angewandte Mathematik

Abstract

In this thesis we examine and optimize backward proof search for two widely used propositional non-monotonic logics: circumscription and default logic. While only one calculus is taken into account for circumscription, we examine two calculi for default logic, one for skeptical the other for credulous entailment. Since the two calculi for default logic depend on a residue calculus — a calculus for the justification-free fragment of default logic — we also examine proof search for this fragment.

We show for all four calculi that simple provers based on backward rule application inherit certain redundancies. For these redundancies we develop algorithms to detect and prevent them.

For the circumscription, residue and skeptical default calculus we successfully adapt the use-check method from proof search in classical propositional logic. This method aims at deducing the provability of one premise of a branching rule by analyzing the proof found for the other premise. If the search tree can be successfully reduced the proving process can be sped up considerably.
We furthermore show how to encode use-check into the different calculi by providing modified versions of the circumscription and residue calculus with inbuilt use-check.

To compare the different optimization techniques we test our algorithms on several scalable problems and also compare the backward proof search algorithms with other approaches. For circumscription we compare them to the approach of expressing circumscription in classical logic while for default logic we compare them to an extension based proving technique.

The developed optimization techniques lead to good results for certain problems while having little or no impact on others. In general, their impact on the circumscription prover is higher than on provers for default logic and use-check proves to be the most valuable optimization technique.

Acknowledgments

I thank my supervisor Gerhard Jäger to have given me the opportunity to write this thesis, for his patience, ideas and support, Jürg Kohlas for refereeing it, and the Swiss National Science Foundation for their financial support.

Heartfelt thanks go to Kai Brünnler for proof reading and correcting my work, for many fruitful discussions that led to new ideas and clarifications, for his moral support and most of all for his friendship.

Writing a thesis always goes with having too little time for your family and friends. I thank them all for being there when in need and for their understanding in times when I made myself rare. My deepest thanks go to Anuschka who kept me grounded with her love and her encouraging and charming way during the most intensive times.

Last but not least, my thanks also go to the current and former members of the logic and theory group at the University of Bern, especially to Alain Heuerding and Stefan Schwendimann for initiating the Logics Workbench in which I have implemented my algorithms.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

In this thesis we inspect proof search for propositional circumscription and credulous and skeptical propositional default logic. We identify redundancies in proof search and give methods to avoid them. This results in doing less backtracking and reducing the search trees. One of these methods is called use-check. When this thesis started, use-check was successfully implemented for classical propositional logic in the logics workbench LWB. Because use-check is designed for monotonic logic, the main issues of this thesis is to adapt it to non-monotonic logics or to develop other methods that take up the idea of use-check.

The calculi we use for our proof search have been developed by Bonatti and Olivetti [2] [3] [4]. They all rely on two calculi for classical logic, the one deduces valid the other invalid sequents. Furthermore the two calculi for default logic rely on a residue calculus, which operates over residue sequents, these are sequents that contain formulas and justification-free defaults.

We take on these calculi and modify some of them to include use-check, namely the circumscription calculus and the residue calculus. In order that these modified calculi obtain the necessary use-check information from the calculi they rely on, we furthermore give a sequent calculus for classical default logic with inbuilt use-check.

For credulous default logic use-check is not applicable since the two rules that process proper defaults do not branch and use-check aims at omitting such branches. For this calculus we use a preprocessing step to limit the search tree and obtain like this a method that is in the sense of use-check.

For skeptical default logic use-check is again applicable. However, for this calculus we do without a modified calculus that does use-check but show how to apply and implement use-check.

The algorithms presented in the thesis are implemented in the LWB. We test our algorithms on some selected scalable formulas to compare the different optimization methods discussed in this thesis.

Outline

In Chapter 1 we introduce classical propositional logic together with the general notion of a sequent calculus. We furthermore give a well known Gentzen-style sequent-calculus for classical logic together with the corresponding refutation calculus from Bonatti and Olivetti [4] which is used in their calculi for propositional circumscription and propositional default logic.

In Chapter 2 we discuss proof search in classical propositional logic and show up optimization techniques. Among them is the well known method called use-check which allows us to cut down the search tree. We sketch a proof search algorithm with use-check and show how to encode use-check into the calculus given in the previous chapter. This encoding serves us as an archetype to encode use-check in further calculi. At the end of the chapter we show the impact of use-check on the pigeonhole formula and the impact of allowing negation on non-atoms on Urquhart's formula [26].

In Chapter 3 we introduce the semantic definition of circumscription on the notion of minimal models. Then we inspect the syntactic definition of propositional circumscription logic which expresses circumscription in a single propositional formula and thus offers the possibility to use a prover for classical proposition logic to validate whether a formula is minimally entailed by a theory. Finally we give the calculus for propositional circumscription logic from Bonatti and Olivetti [2].

Chapter 4 covers automatic proving in propositional circumscription. We discuss two approaches to prove minimal entailment. The first approach uses the syntactic definition of circumscription logic. We show some enhancements that can be of use to reduce the resulting circumscription theory. The second approach is based on backward proof search. We start with a simple prover and point out some easy methods to omit redundant classical proofs and refutations. Then we take up the idea of use-check for our prover and identify the necessary conditions to predict for branching rules the provability of one premise from the information gained in the proof of the other premise. From this result we then create a sequent calculus that includes use-check by encoding the necessary information we need for use-check into the deduction rules. Finally we evaluate our optimizations on some scalable problems. There we compared three main approaches. The first is to use the syntactic definition of circumscription logic, the second is to use a prover that backward applies the deduction rules and prefers branching to non-branching rules, the third is the complement of the second prover, i.e. one that prefers non-branching to branching rules. We compare our provers to mm [22], a prover that is restricted to sets of clauses.

In Chapter 5 we introduce propositional default logic with some well known examples. Then we recall the calculi of Bonatti and Olivetti for credulous and skeptical default logic [3] together with two calculi for the justification-free fragment of default logic.

In Chapter 6 we investigate two approaches to prove a residue sequent.
The first approach is based on backward proof search. We start with a simple prover, identify redundancies in the proof search tree and show how to avoid them. This leads to the optimization we have named general improvement. The name is derived from the fact that the method depends only on the position on nodes in the search tree and not on the sequent that is to prove. The implementation of general improvement is defined on a partial relation on paths in the search tree. We show that if a path is in relation to another then the nodes encountered at the corresponding positions are in a subsequent relation. This allowed us to implement general improvement on comparing two positions in the search tree instead of comparing two sequents. Then we develop use-check for the residue sequent calculus. We first present a calculus that encodes use-check and derive from it the corresponding algorithm. Finally we show how to combine general improvement with use-check.
The second approach is based on calculating the closure of the residue theory. Since the closure operator defines an infinite set of formulas we first show how to compute a finite set of formulas that defines the closure. In a further step we then aim at computing only a partial closure that contains the formula that is to prove. To do so we introduced so called (minimal) quasi-supports with the intention to extend them to supports, these are sets of residues that correspond to closed subtheories of the closure. Before discussing the calculation of minimal quasi-supports we first present a proving algorithm that bases on them and point out some redundancies which can be resolved by caching intermediate results. Then we turn towards the problem of computing the minimal quasi-supports. The approach we choose is based on a modified version of the use-check calculus for classical logic. By extending the sequents with additional information to distinct formulas in the base theory from formulas that derive from residues, we succeed in computing minimal quasi-supports based on the modified calculus.
At the end of the chapter we evaluate our algorithms and optimization techniques on some scalable problems. There we investigate four kind of provers. Two of them do backward proof search and prefer to either apply the non-branching rule or the branching rule first. The other two base on calculating the closure. One of them calculates the full closure while the other follows the approach based on the quasi-supports.

In Chapter 7 we investigate backward proof search in default logic.
We start with a simple prover of the calculus for credulous entailment, identify obvious redundancies of that prover and show that they can be avoided

if we base the prover on partitioning the set of defaults instead of backward applying the deduction rules. In a next step we introduce a preprocessing step that may lead to a reduced proof search. This preprocessing is based on computing the so called minimal supports of a default theory. We show how the minimal supports can be computed from the minimal quasi-supports. Then we point out that we can use information gained in the proof of residue sequents to predict the validity of yet unprocessed residue sequents. This leads again to a method which caches intermediate results to speed up proof search.
Then we continue with a simple prover of the calculus for skeptical entailment. There we have again redundancies that can be avoided if we base the prover on partitioning the set of defaults instead of backward applying the deduction rules. Then we show as in the credulous case that we can reuse intermediate results gained in the proof of residue sequents to predict the validity or invalidity of yet unprocessed residue sequents. Finally we develop use-check for the skeptical prover.
To have a comparison with an other approach we then also develop an algorithm to compute the extensions of a default theory. It is a refinement of an algorithm given by Marek and Truszczyński [19].
We close the chapter by comparing our optimizations and approaches on some scalable problems.

Chapter 1

Classical Propositional Logic

Classical propositional logic is needed for the definition and the calculi of the two non-monotonic logics presented later in this thesis. In this chapter we give the syntactic and semantic definitions of classical propositional logic and an abstract notion of a sequent calculus. We close the chapter with the definition of two sequent calculi for classical propositional logic: the classical propositional calculus CPC and the classical propositional refutation calculus CPRC.

1.1 Syntax

In this section we define the language $\mathcal{L}$ of propositional logic and some syntactic abbreviations and properties used in all logics. Furthermore we define some notation conventions, e.g. what symbols we use to represent formulas or theories.

1.1.1 Core Language

Definition 1.1 (language of propositional logic)
The *language* $\mathcal{L}(\mathcal{V})$ of propositional logic is based on a countable set $\mathcal{V}$ of propositional variables, the truth symbol $\top$ (true), the connectives $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction), and the parentheses '(' and ')'.

The elements of $\mathcal{L}(\mathcal{V})$ are called *formulas* and defined inductively as follows:

1. $\top \in \mathcal{L}(\mathcal{V})$ and $p \in \mathcal{L}(\mathcal{V})$ for all $p \in \mathcal{V}$.
2. $(\neg\top) \in \mathcal{L}(\mathcal{V})$ and $(\neg p) \in \mathcal{L}(\mathcal{V})$ for all $p \in \mathcal{V}$.

3. If $A \in \mathcal{L}(\mathcal{V})$ and $B \in \mathcal{L}(\mathcal{V})$ then $(A \wedge B) \in \mathcal{L}(\mathcal{V})$ and $(A \vee B) \in \mathcal{L}(\mathcal{V})$.

Formulas of the first form are called *atomic*.

A formula A is called a *literal* if it is of the first or second form. We distinguish between positive $(p, \top)$ and negative $(\neg p, \neg\top)$ literals.

Given a set of variables P we sometimes write $\neg P$ to denote $\{\neg p : p \in P\}$.

1.1.2 Meta Symbols

In unambiguous situations we normally write $\mathcal{L}$ instead of $\mathcal{L}(\mathcal{V})$.

We use the symbols p, q, r, s, t to denote propositional variables and A, B, C, D to denote formulas. The symbols may be subscripted.

1.1.3 Syntactic Abbreviations

To keep $\mathcal{L}(\mathcal{V})$ small, the connectives for implication $(\rightarrow)$ and equation $(\leftrightarrow)$ are not part of the languages and negation is currently only defined on atomic formulas. To use those constructs anyway, we define them in the usual way.

We drop double negation on atomic formulas and define $\bot$ to be an abbreviation of $\neg\top$.

Definition 1.2 ($\bot$ and double negations)

$$(\neg(\neg p)) := p \qquad (\neg(\neg\top)) := \top \qquad \bot := (\neg\top)$$

Using De Morgan's rules we inductively define negation on a disjunction and conjunction and can thus define implication and equation on arbitrary formulas.

Definition 1.3 (negation on non-variables, $\rightarrow$ and $\leftrightarrow$)

$$(\neg(A \wedge B)) := ((\neg A) \vee (\neg B)) \quad (\neg(A \vee B)) := ((\neg A) \wedge (\neg B))$$
$$(A \rightarrow B) := ((\neg A) \vee (B)) \qquad (A \leftrightarrow B) := ((A \rightarrow B) \wedge (B \rightarrow A))$$

1.1.4 Precedence of Connectives

Outer parentheses in formulas are normally omitted, e.g. we write $A \wedge B$ instead of $(A \wedge B)$. To omit further parentheses we define the following

precedence for the connectives: $\neg \ < \ \wedge \ < \ \vee \ < \rightarrow < \leftrightarrow$, so

$$\neg p \vee q \wedge r \leftrightarrow s \rightarrow q = ((\neg p) \vee (q \wedge r)) \leftrightarrow (s \rightarrow q).$$

Furthermore we define $\vee, \wedge$ and $\leftrightarrow$ to associate to the left and $\rightarrow$ to associate to the right, so

$$p_1 \vee p_2 \vee p_3 = (p_1 \vee p_2) \vee p_3 \qquad p_1 \rightarrow p_2 \rightarrow p_3 = p_1 \rightarrow (p_2 \rightarrow p_3).$$

Given a set or multiset of formulas $\Gamma = \{A_1, \ldots, A_n\}$ we use the following abbreviations:

$$\bigvee \Gamma := A_1 \vee A_2 \vee \ldots \vee A_n \qquad \bigwedge \Gamma := A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

Remark 1.4 (multisets of formulas)
For proof search we use calculi without weakening rules. Because of this we sometimes need multiple occurrences of formulas in a sequent and hence use multisets instead of sets in the sequents of these calculi.

1.1.5 Further Syntactic Definitions

Definition 1.5 (substitution)
For $A, B_0, B_1, \ldots, B_n \in \mathcal{L}$ we write $A[B_0/p_0, B_1/p_1, \ldots, B_n/p_n]$ to denote the formula in which all occurrences of p_i are simultaneously replaced by B_i.

Definition 1.6 (formula length, subformulas, variables)
Let $\star$ denote any of the binary connectives $\vee$ or $\wedge$.

The *length*$\mathrm{len}(A)$ of a formula $A \in \mathcal{L}$ is defined inductively as follows:

$$\begin{array}{rr} \mathrm{len}(p) := 1 & \mathrm{len}(\neg A) := \mathrm{len}(A) + 1 \\ \mathrm{len}(\top) := 1 & \mathrm{len}(A \star B) := \mathrm{len}(A) + \mathrm{len}(B) + 1 \end{array}$$

The multiset of *subformulas* $\mathrm{sub}(A)$ of a formula $A \in \mathcal{L}$ is defined inductively as follows:

$$\begin{array}{rr} \mathrm{sub}(p) := \{p\} & \mathrm{sub}(\neg A) := \mathrm{sub}(A) \cup \{\neg A\} \\ \mathrm{sub}(\top) := \{\top\} & \mathrm{sub}(A \star B) := \mathrm{sub}(A) \cup \mathrm{sub}(B) \cup \{A \star B\} \end{array}$$

The set of *variables* $\mathrm{vars}(A)$ of a formula $A \in \mathcal{L}$ is defined inductively as follows:

$$\begin{array}{rr} \mathrm{vars}(p) := \{p\} & \mathrm{vars}(\neg A) := \mathrm{vars}(A) \\ \mathrm{vars}(\top) := \emptyset & \mathrm{vars}(A \star B) := \mathrm{vars}(A) \cup \mathrm{vars}(B) \end{array}$$

1.2 Semantics

The language of propositional logic as given above is used for many logics, which all differ by their semantical interpretations. Classical propositional logic (CPL) is one of them. In this section we introduce the semantics of CPL together with some definitions and properties.

Definition 1.7 (interpretation, validity, model)
An *interpretation* I of $\mathcal{L}(\mathcal{V})$ is a mapping $I : \mathcal{V} \to \{0, 1\}$.

The *extension* $\hat{I} : \mathcal{L} \to \{0, 1\}$ of an interpretation I is defined inductively as follows:

$$\hat{I}(\top) := 1 \qquad \hat{I}(\neg\top) := 0 \qquad \hat{I}(p) := I(p) \qquad \hat{I}(\neg p) := 1 - I(p)$$

$$\hat{I}(A \vee B) := \begin{cases} 0 & \text{if } \hat{I}(A) = 0 \text{ and } \hat{I}(B) = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\hat{I}(A \wedge B) := \begin{cases} 1 & \text{if } \hat{I}(A) = 1 \text{ and } \hat{I}(B) = 1 \\ 0 & \text{otherwise} \end{cases}$$

A formula $A \in \mathcal{L}$ is *valid in* I if $\hat{I}(A) = 1$. We then write $I \vDash A$ and call I a *model of* A.
Accordingly we define a set of formulas $T \subseteq \mathcal{L}$ to be *valid in* I if $I \vDash A$ for all $A \in T$. I is then also called a *model of* T.

A formula $A \in \mathcal{L}$ is a *logical consequence* of a set of formulas T if $I \vDash A$ for all models I of T. We then write $T \Vdash A$. Furthermore we define $\mathrm{Th}(T) := \{A \in \mathcal{L} : T \Vdash A\}$ to be the set of all logical consequences of T.

If a formula $A \in \mathcal{L}$ is valid in all interpretations of $\mathcal{L}$ we call it a *tautology* and write $\vDash A$.

Notation 1.8 (symbols used for interpretations, $\nvDash$)
We use the symbols I, J, M and N for interpretations.
Furthermore we write $I \nvDash A$ if A is not valid in I, $I \nvDash T$ if I is not a model of T, $T \nVdash A$ if A is not a logical consequence of T and $\nvDash A$ if A is not a tautology.

We sometimes define an interpretation through the set of variables for which it returns 1, i.e. we write $I := \{p_1, p_2, \ldots, p_n\}$ to define an interpretation I that maps p_i to 1 for $1 \leq i \leq n$ and all other propositional variables to 0.

Lemma 1.9 (properties of $\vDash$)
Let I be an interpretation and A and B be formulas.

1. $I \vDash A$ if and only if $I \nvDash \neg A$.

2. If $I \vDash A$ then $I \vDash A \vee B$.
3. If $I \vDash A \vee B$ then $I \vDash B \vee A$.
4. $I \vDash A$ and $I \vDash B$ if and only if $I \vDash A \wedge B$.

1.3 General Notion of a Sequent Calculus

The proof search algorithms for classical propositional logic in the LWB are based on two-sided sequent calculi, also called Gentzen-style calculi. In this section we give some general definitions which are common to each individual calculus used in this work.
Since the sequents used in a calculus depend on the calculus itself, we use the word 'sequent' as a placeholder for a term which is to be defined when giving the definition of the specific calculus.

1.3.1 Deduction Rules

Definition 1.10 (deduction rule, sequent calculus)
A *deduction rule* R is a pair $\langle \mathcal{S}, \mathrm{S} \rangle$ where $\mathcal{S} = \{\mathrm{S}_1, \ldots, \mathrm{S}_n\}$ is a possibly empty finite set of sequents schemes called *premises* and S is a sequent scheme called *conclusion*. Formally we write

$$\frac{\mathrm{S}_1 \quad \mathrm{S}_2 \quad \ldots \quad \mathrm{S}_n}{\mathrm{S}}(\mathrm{R}).$$

Deduction rules without premises are called *axioms*, the others are called *proper rules*.

A finite set of deduction rules $\mathcal{C} = \{\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_n\}$ is called a *sequent calculus*.

Deduction rules consist of *sequent schemes*. These are abstract sequents that may contain variables for different kinds of objects (e.g. formulas, formula sets, natural numbers). Sequents are obtained from sequent schemes by instantiating them, which means that all variables in the sequent scheme are substituted with corresponding objects. If we have for example a sequent scheme of the form $A \supset A \vee B$ then a possible instance might be $p_0 \vee p_1 \supset (p_0 \vee p_1) \vee p_2$, provided that p_0, p_1 and p_2 are in the set of variables of our language.

Accordingly a *rule instance*, as used in proofs, is obtained from a deduction rule R by instantiating its sequents. A deduction rule thus stands for all rule instances that can be obtained from it.

In the following we often speak of a deduction rule or rule when referring to an instance of it.

Example 1.11 (deduction rule as rule instances)
To illustrate the remark above, we give two examples for a sequent calculus for the language $\mathcal{L}(\mathcal{V})$. For clarity the rule instances are written without rule name.

$$\begin{aligned} \frac{\neg\neg p}{p}(\neg\neg) &= \left\{ \frac{\neg\neg p}{p} : p \in \mathcal{V} \right\} \\ \frac{A \qquad B}{A \wedge B}(\wedge) &= \left\{ \frac{A \qquad B}{A \wedge B} : A \in \mathcal{L}, B \in \mathcal{L} \right\} \end{aligned}$$

1.3.2 Proofs

Definition 1.12 (deducible sequent)
Let $\mathcal{C}$ be a sequent calculus. We then define a sequent S to be *deducible in* $\mathcal{C}$ (formally $\mathcal{C} \vdash \mathrm{S}$) as follows:

1. If S is the conclusion of an axiom of $\mathcal{C}$ then $\mathcal{C} \vdash \mathrm{S}$.
2. If $\mathrm{S}_1, \ldots, \mathrm{S}_n$ are the premises of a deduction rule of $\mathcal{C}$ that has S as its conclusion and $\mathcal{C} \vdash \mathrm{S}_i$ for all $1 \leq i \leq n$ then $\mathcal{C} \vdash \mathrm{S}$.

A deduction of a sequent S has the form of a tree. The axioms thereby represent the leaves of the tree and the resulting sequent the root of it. Such a deduction tree of a sequent S with respect to $\mathcal{C}$ is called a *proof of* S *in* $\mathcal{C}$.

Often we prove properties using the depth of a proof, which, roughly said, is the number of proper rules the longest branch in the proof contains. The precise definition is given below.

Definition 1.13 (depth of a proof)
Let $\mathcal{P}$ be a proof of a sequent S, R be the last deduction rule of $\mathcal{P}$ having the premises $\mathcal{S} := \{\mathrm{S}_1, \ldots, \mathrm{S}_n\}$ and the conclusion S, and $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be the proofs of $\mathrm{S}_1, \ldots, \mathrm{S}_n$ in $\mathcal{P}$.
Then the *depth* $\mathrm{depth}(\mathcal{P})$ of $\mathcal{P}$ is defined inductively as follows:

$$\mathrm{depth}(\mathcal{P}) := \begin{cases} 0 & \text{if } \mathcal{S} \text{ is empty,} \\ \max(\mathrm{depth}(\mathcal{P}_1), \ldots, \mathrm{depth}(\mathcal{P}_n)) + 1 & \text{otherwise.} \end{cases}$$

When doing proof search using backward rule application we sometimes need to backtrack in the algorithm. Some rules have the nice property that they are invertible and no backtracking is necessary when applying those rules backwards.

Definition 1.14 (invertible rules)
Let R be a proper rule of a calculus $\mathcal{C}$ having $\mathcal{S} := \{\mathrm{S}_1, \ldots, \mathrm{S}_n\}$ as premises and S as conclusion. Then R is called *invertible* if the following holds:

$$\text{If} \quad \mathcal{C} \vdash \mathrm{S} \quad \text{then} \quad \mathcal{C} \vdash \mathrm{S}_i, \text{ for all } \mathrm{S}_i \in \mathcal{S}$$

1.4 Classical Propositional Calculus

In this section we introduce a two-sided sequent calculus CPC for classical propositional logic. Since the calculus is known to be complete, we omit soundness and completeness proofs. In the next chapter we use the calculus to introduce proof search and an optimization technique called "use-check".

1.4.1 Classical Sequents

Definition 1.15 (CPC sequent)
A CPC *sequent* is a pair of multisets of formulas $\langle \Gamma, \Delta \rangle$ denoted by $\Gamma \supset \Delta$. We call Γ the *antecedent* and Δ the *succedent* of the sequent.

Definition 1.16 (length of a sequent)
The *length* of a sequent $\Gamma \supset \Delta$ is defined to be the sum of the length of its formulas.

$$\mathrm{len}(\Gamma \supset \Delta) := \sum_{A \in \Gamma \cup \Delta} \mathrm{len}(A)$$

Since Γ and Δ are multisets their union $\Gamma \cup \Delta$ is to be understood as multiset-union, i.e. multiple occurrences of formulas are relevant for this definition.

Definition 1.17 (valid sequent)
A sequent $\Gamma \supset \Delta$ is defined to be *valid* if $\vDash \bigwedge \Gamma \to \bigvee \Delta$. We write $\vDash \Gamma \supset \Delta$ to denote the validity of a sequent.

Remark 1.18 (countermodel)
Given a sequent $\Gamma \supset \Delta$ we call an interpretation M a *countermodel* of $\Gamma \supset \Delta$ if $M \vDash \Gamma$ and $M \nvDash \bigvee \Delta$.

A sequent has a countermodel if and only if it is not valid.

If $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$ and M is a countermodel of $\Gamma \supset \Delta$ then M is also a countermodel of $\Gamma' \supset \Delta'$.

Proposition 1.19 (monotonicity)
If $\vDash \Gamma \supset \Delta$ then $\vDash \Gamma, \Gamma' \supset \Delta, \Delta'$.

$$\frac{}{\mathbf{p}, \Gamma \supset \mathbf{p}, \Delta}(\mathsf{id}) \qquad \frac{}{\Gamma \supset \top, \Delta}(\top)$$

$$\frac{\Gamma \supset \Delta, A}{\Gamma, \neg\mathbf{A} \supset \Delta}(\neg\supset) \qquad \frac{\Gamma, A \supset \Delta}{\Gamma \supset \Delta, \neg\mathbf{A}}(\supset\neg)$$

$$\frac{\Gamma, A, B \supset \Delta}{\Gamma, \mathbf{A} \wedge \mathbf{B} \supset \Delta}(\wedge\supset) \qquad \frac{\Gamma \supset \Delta, A \qquad \Gamma \supset \Delta, B}{\Gamma \supset \Delta, \mathbf{A} \wedge \mathbf{B}}(\supset\wedge)$$

$$\frac{\Gamma, A \supset \Delta \qquad \Gamma, B \supset \Delta}{\Gamma, \mathbf{A} \vee \mathbf{B} \supset \Delta}(\vee\supset) \qquad \frac{\Gamma \supset \Delta, A, B}{\Gamma \supset \Delta, \mathbf{A} \vee \mathbf{B}}(\supset\vee)$$

Figure 1.1: Deduction rules of CPC

1.4.2 The Rules of the Classical Propositional Calculus

We are now ready to give the sequent calculus CPC for classical propositional logic. The calculus is cut-free and thus well suited for proof search algorithms which are based on backward application of the rules of a calculus. Since we are using a two-sided calculus, we have rules for negations on non-atomic formulas. To have more compact proof representations we furthermore introduce macro-rules for the defined connectives $\rightarrow$ and $\leftrightarrow$.

Definition 1.20 (CPC)
We define the classical propositional sequent calculus CPC to have the deduction rules as given in Figure 1.1.

$\top, \mathbf{p}, \neg\mathbf{A}, \mathbf{A} \wedge \mathbf{B}$ and $\mathbf{A} \vee \mathbf{B}$ are called *principal formulas*, A and B are called *active formulas in the premise*.

It is well known that CPC is sound and complete, we therefore omit the proof of the corresponding theorem.

Theorem 1.21 (soundness and completeness of CPC)

$$\mathsf{CPC} \vdash \Gamma \supset \Delta \quad \text{iff} \quad \models \Gamma \supset \Delta.$$

Proposition 1.22 (invertibility of CPC)
The proper rules of CPC are invertible.

Proof. We only show that the rule $(\vee\supset)$ is invertible and leave it up to the reader to verify that the other rules are invertible, too.
Suppose that $\Gamma, A \vee B \supset \Delta$ is deducible in CPC and let $\mathcal{P}$ be a proof of $\Gamma, A \vee B \supset \Delta$. We show our claim by induction on $d := \mathrm{depth}(P)$.

- $d = 0$: Then $\Gamma, A \vee B \supset \Delta$ is an axiom and hence $\Gamma, A \supset \Delta$ and $\Gamma, B \supset \Delta$ are axioms, too.

- $d > 0$: We make a case distinction on the last rule of $\mathcal{P}$:
  - $\dfrac{\Gamma, A \supset \Delta \qquad \Gamma, B \supset \Delta}{\Gamma, A \vee B \supset \Delta}(\vee\supset)$

    Then trivially $\Gamma, A \supset \Delta$ and $\Gamma, B \supset \Delta$ are deducible in CPC.
  - $\dfrac{\Gamma, A \vee B \supset \Delta', C \qquad \Gamma, A \vee B \supset \Delta', D}{\Gamma, A \vee B \supset \underbrace{\Delta', C \wedge D}_{\Delta}}(\supset\wedge)$

    By induction hypothesis we know that the following four sequents are deducible in CPC.

    $$\Gamma, A \supset \Delta', C \quad (1) \qquad \Gamma, B \supset \Delta', C \quad (2)$$
    $$\Gamma, A \supset \Delta', D \quad (3) \qquad \Gamma, B \supset \Delta', D \quad (4)$$

    With $(\supset \wedge)$ we can hence deduce $\Gamma, A \supset \Delta', C \wedge D$ from (1) and (3) and $\Gamma, B \supset \Delta', C \wedge D$ from (2) and (4).

  The other cases are similar to the last case.

□

Since we are using multisets in our sequents we do not have automatic contraction. According to the following lemma we know that a contraction rule is not needed.

Lemma 1.23 (contraction)
1. If $\mathsf{CPC} \vdash \Gamma \supset \Delta, A, A$ then $\mathsf{CPC} \vdash \Gamma \supset \Delta, A$.
2. If $\mathsf{CPC} \vdash \Gamma, A, A \supset \Delta$ then $\mathsf{CPC} \vdash \Gamma, A \supset \Delta$.

Proof. We show the two claims simultaneously by induction on $\mathrm{len}(A)$.
Let $\mathcal{P}$ be a CPC-proof of $\Gamma \supset \Delta, A, A$ (contraction in the antecedent is shown accordingly).

$\mathrm{len}(A) = 1$.

Then $A := p$ or $A := \top$. We only show the first case.

- $A := p$. Since p is atomic, any sequent in $\mathcal{P}$ is of the form $\Gamma' \supset \Delta', p, p$ and p does not occur as principal formula in any structural rule of $\mathcal{P}$. If p is used as principal formula in an axiom $\Gamma', \mathbf{p} \supset \Delta, \mathbf{p}, p$ of $\mathcal{P}$, then $\Gamma', \mathbf{p} \supset \Delta, \mathbf{p}$ is an axiom, too. We thus obtain a proof $\mathcal{P}'$ of $\Gamma \supset \Delta, p$ from $\mathcal{P}$ if in every antecedent of $\mathcal{P}$ one occurrence of p is removed.

$\mathrm{len}(A) > 1$.

We make a case distinction on the form of A but only show two cases, since the other cases are similar.

- $A := \neg B$
  Since $(\neg \supset)$ is invertible we know that $\Gamma, B \supset \Delta, \neg B$ is provable in CPC. For the same reason we know that $\Gamma, B, B \supset \Delta$ is provable in CPC. By induction hypothesis we furthermore know that $\Gamma, B \supset \Delta$ is provable in CPC. Using $(\neg \supset)$ we can deduce $\Gamma \supset \Delta, \neg B$.

- $A := B \wedge C$
  Since $(\wedge \supset)$ is invertible we know that $\Gamma \supset \Delta, B \wedge C, B$ and $\Gamma \supset \Delta, B \wedge C, C$ are provable in CPC. For the same reason we know that $\Gamma \supset \Delta, B, B$, $\Gamma \supset \Delta, C, B$, $\Gamma \supset \Delta, B, C$, and $\Gamma \supset \Delta, C, C$ are provable in CPC. By induction hypothesis we furthermore know that $\Gamma \supset \Delta, B$ and $\Gamma \supset \Delta, C$ are provable in CPC. Using $(\wedge \supset)$ we can deduce $\Gamma \supset \Delta, B \wedge C$.

□

Macro-Rules

For efficiency reasons we operate for proof search over an extended language $\mathcal{L}^+$ in which the connectives $\rightarrow$ and $\leftrightarrow$ are part of the language. An extension of an interpretation I of $\mathcal{L}^+(\mathcal{V})$ is defined straightforward, i.e. $\hat{I}(A \rightarrow B) = \hat{I}(\neg A \vee B)$ and $\hat{I}(A \leftrightarrow B) = \hat{I}((A \rightarrow B) \wedge (B \rightarrow A))$.

For $\mathcal{L}^+$ we need an extended calculus CPC^+ that contains the following additional rules.

$$\frac{\Gamma \supset \Delta, A \qquad \Gamma, B \supset \Delta}{\Gamma, \mathbf{A \rightarrow B} \supset \Delta}(\rightarrow\supset) \qquad \frac{\Gamma, A \supset \Delta, B}{\Gamma \supset \Delta, \mathbf{A \rightarrow B}}(\supset\rightarrow)$$

$$\frac{\Gamma, A, B \supset \Delta \qquad \Gamma \supset \Delta, A, B}{\Gamma, \mathbf{A \leftrightarrow B} \supset \Delta}(\leftrightarrow\supset) \qquad \frac{\Gamma, A \supset \Delta, B \qquad \Gamma, B \supset \Delta, A}{\Gamma \supset \Delta, \mathbf{A \leftrightarrow B}}(\supset\leftrightarrow)$$

These rules can be seen as macro-rules built from CPC rules. While $(\rightarrow\supset)$, $(\supset\rightarrow)$ and $(\supset\leftrightarrow)$ are pure macro-rules (two examples are given below) the rule $(\leftrightarrow\supset)$ contains an implicit simplification that drops two premises of which we know that they are valid.

$$\frac{\dfrac{\Gamma \supset \Delta, A}{\Gamma, \neg A \supset \Delta}(\neg\supset) \qquad \Gamma, B \supset \Delta}{\Gamma, \underbrace{\neg \mathbf{A} \vee \mathbf{B}}_{A \rightarrow B} \supset \Delta}(\vee\supset) \qquad \frac{\Gamma, A \supset \Delta, B}{\dfrac{\Gamma \supset \Delta, \neg \mathbf{A}, B}{\Gamma \supset \Delta, \underbrace{\neg \mathbf{A} \vee \mathbf{B}}_{A \rightarrow B}}(\supset\vee)}(\supset\neg)$$

$$\dfrac{\dfrac{\Gamma \supset \Delta, A, B \quad \dfrac{\text{valid}}{\Gamma, A \supset \Delta, A}}{\Gamma, \mathbf{B} \to \mathbf{A} \supset \Delta, A}(\to\supset) \quad \dfrac{\dfrac{\text{valid}}{\Gamma, B \supset \Delta, B} \quad \Gamma, B, A \supset \Delta}{\Gamma, B, \mathbf{B} \to \mathbf{A} \supset \Delta}(\to\supset)}{\dfrac{\Gamma, \mathbf{A} \to \mathbf{B}, B \to A \supset \Delta}{\Gamma, \underbrace{(\mathbf{A} \to \mathbf{B}) \wedge (\mathbf{B} \to \mathbf{A})}_{A \leftrightarrow B} \supset \Delta}(\wedge\supset)}(\to\supset)$$

Since the macro rules only reflect applications of several core rules, the results about soundness and completeness of CPC are not affected when adding those rules to it. Furthermore, since all rules of CPC are invertible, the macro rules that do not contain an implicit simplification also have that property. That this is also the case for the rule $(\leftrightarrow\supset)$ can easily be verified.

1.5 Classical Refutation Calculus

The calculi we use for propositional circumscription and propositional default logic make use of a classical propositional calculus and a refutation calculus for classical propositional logic developed by Bonatti [5]. The refutation calculus is converse to the sequent prover in that it is used to show non-provability.
In this section we introduce this calculus and close with some examples.

In contrast to the definitions in the paper of Bonatti we speak of sequents instead of anti-sequents. As a consequence we speak of refuted sequents instead of provable anti-sequents and therefore renamed Bonatti's calculus from 'anti-sequent calculus' to 'refutation calculus'. The rules themselves are left unchanged.

1.5.1 The Rules of the Classical Refutation Calculus

Below we give the definition of the refutation calculus CPRC. As in CPC we have rules for negation on non-atomic formulas and have macro rules for the defined connectives $\to$ and $\leftrightarrow$. Since CPRC is not a standard calculus we give the proofs for soundness and completeness of it, i.e. we show that if a sequent is refutable in CPRC then it is not valid and if a sequent is not valid, then it is refutable in CPRC.

Definition 1.24 (the calculus CPRC)
We define the classical propositional refutation calculus CPRC to have the deduction rules as given in Figure 1.2.

Since we use the calculus to refute a sequent, the rule (aax) is called an *anti-axiom*.

$$\frac{}{\Gamma \supset \Delta}(\mathsf{aax}) \text{ with } \Gamma \subseteq \mathcal{V} \cup \{\top\}, \Delta \subseteq \mathcal{V} \text{ and } \Gamma \cap \Delta = \emptyset$$

$$\frac{\Gamma \supset \Delta, A}{\Gamma, \neg \mathbf{A} \supset \Delta}(\neg\not\supset) \qquad \frac{\Gamma, A \supset \Delta}{\Gamma \supset \Delta, \neg \mathbf{A}}(\not\supset\neg)$$

$$\frac{\Gamma, A, B \supset \Delta}{\Gamma, \mathbf{A} \wedge \mathbf{B} \supset \Delta}(\wedge\not\supset) \qquad \frac{\Gamma \supset \Delta, A, B}{\Gamma \supset \Delta, \mathbf{A} \vee \mathbf{B}}(\not\supset\vee)$$

$$\frac{\Gamma, A \supset \Delta}{\Gamma, \mathbf{A} \vee \mathbf{B} \supset \Delta}(\cdot\vee\not\supset) \qquad \frac{\Gamma \supset \Delta, A}{\Gamma \supset \Delta, \mathbf{A} \wedge \mathbf{B}}(\not\supset\cdot\wedge)$$

$$\frac{\Gamma, B \supset \Delta}{\Gamma, \mathbf{A} \vee \mathbf{B} \supset \Delta}(\vee\cdot\not\supset) \qquad \frac{\Gamma \supset \Delta, B}{\Gamma \supset \Delta, \mathbf{A} \wedge \mathbf{B}}(\not\supset\wedge\cdot)$$

Figure 1.2: Deduction rules of CPRC

The sequents used in the refutation calculus are equivalent to those of the calculus CPC. If it is unclear from the context, we write $\mathsf{CPC} \vdash \Gamma \supset \Delta$ and $\mathsf{CPRC} \vdash \Gamma \supset \Delta$ to denote that a sequent is provable and refutable, respectively.

Definition 1.25 (refutable sequent, refutation)
A CPC sequent is called *refutable* if it is deducible in CPRC. We thus normally speak of a *refutation* instead of a proof in CPRC.

Theorem 1.26 (soundness)
If a CPC sequent $\Gamma \supset \Delta$ is refutable then it is not valid.

Proof. Suppose we have a refutation $\mathcal{P}$ of a sequent $\Gamma \supset \Delta$ in CPRC. We show by induction on the depth of $\mathcal{P}$, that $\Gamma \supset \Delta$ is not valid.

- $\mathrm{depth}(\mathcal{P}) = 0$.

  Then $\Gamma \supset \Delta$ is an anti-axiom of CPRC, i.e. $\Gamma \subset \mathcal{V} \cup \{\top\}, \Delta \subset \mathcal{V}$ and $\Gamma \cap \Delta = \emptyset$. Let $M := \Gamma \cap \mathcal{V}$, then $M \vDash \Gamma$, $M \vDash \top$ and (since $\Gamma \cap \Delta = \emptyset$) $M \nvDash p$ for all $p \in \Delta$, thus $\Gamma \supset \Delta$ is not valid.

- $\mathrm{depth}(\mathcal{P}) = n + 1$.

  Suppose the last step of the refutation is:

  – $\dfrac{\Gamma' \supset \Delta, A}{\Gamma', \neg A \supset \Delta}(\neg\not\supset)$

    By induction hypothesis $\Gamma' \supset \Delta, A$ is not valid. Let M be a countermodel of it. Then $M \nvDash A$ and by Lemma 1.9 we know $M \vDash \neg A$. Hence M is also a countermodel of $\Gamma', \neg A \supset \Delta$ which is thus not valid.

– $\dfrac{\Gamma, A \supset \Delta'}{\Gamma \supset \Delta', \neg A}(\not\supset\neg)$

By induction hypothesis $\Gamma, A \supset \Delta'$ is not valid. Let M be a countermodel of it. Then $M \vDash A$ and by Lemma 1.9 we know $M \nvDash \neg A$, furthermore M is also a model of Γ. Hence M is also a countermodel of $\Gamma \supset \Delta', \neg A$ which is thus not valid.

– $\dfrac{\Gamma', A, B \supset \Delta}{\Gamma', A \wedge B \supset \Delta}(\wedge\not\supset)$

By induction hypothesis $\Gamma', A, B \supset \Delta$ is not valid. Let M be a countermodel of it. Then $M \vDash A$ and $M \vDash B$ and by Lemma 1.9 we know $M \vDash A \wedge B$. Hence M is also a countermodel of $\Gamma', A \wedge B \supset \Delta$ which is thus not valid.

– $\dfrac{\Gamma \supset \Delta', A}{\Gamma \supset \Delta', A \wedge B}(\not\supset\cdot\wedge)$

By induction hypothesis $\Gamma \supset \Delta', A$ is not valid. Let M be a countermodel of it. Then $M \nvDash A$ and by Lemma 1.9 we know $M \nvDash A \wedge B$ for any formula B. Hence M is also a countermodel of $\Gamma \supset \Delta', A \wedge B$ which is thus not valid.

– $\dfrac{\Gamma \supset \Delta', B}{\Gamma \supset \Delta', A \wedge B}(\not\supset\wedge\cdot)$

Analogous to the previous case.

– $\dfrac{\Gamma', A \supset \Delta}{\Gamma', A \vee B \supset \Delta}(\cdot\vee\not\supset)$

By induction hypothesis $\Gamma', A \supset \Delta$ is not valid. Let M be a countermodel of it. Then $M \vDash A$ and by Lemma 1.9 we know $M \vDash A \vee B$ for any formula B. Hence M is also a countermodel of $\Gamma', A \vee B \supset \Delta$ which is thus not valid.

– $\dfrac{\Gamma', B \supset \Delta}{\Gamma', A \vee B \supset \Delta}(\vee\cdot\not\supset)$

Analog to the previous case.

– $\dfrac{\Gamma \supset \Delta', A, B}{\Gamma \supset \Delta', A \vee B}(\not\supset\vee)$

By induction hypothesis $\Gamma \supset \Delta', A, B$ is not valid. Let M be a countermodel of it. Then $M \nvDash A$ and $M \nvDash B$ and by Lemma 1.9 we know $M \nvDash A \vee B$. Hence M is also a countermodel of $\Gamma \supset \Delta', A \vee B$ which is thus not valid.

□

Lemma 1.27 (properties of $\nvDash$)
Let $\Gamma \supset \Delta$ be a CPC sequent and A and B be formulas.

1. If $\nvDash \Gamma, \neg A \supset \Delta$ then $\nvDash \Gamma \supset \Delta, A$.
2. If $\nvDash \Gamma \supset \Delta, \neg A$ then $\nvDash \Gamma, A \supset \Delta$.
3. If $\nvDash \Gamma, A \wedge B \supset \Delta$ then $\nvDash \Gamma, A, B \supset \Delta$.
4. If $\nvDash \Gamma \supset \Delta, A \wedge B$ then $\nvDash \Gamma \supset \Delta, A$ or $\nvDash \Gamma \supset \Delta, B$.
5. If $\nvDash \Gamma, A \vee B \supset \Delta$ then $\nvDash \Gamma, A \supset \Delta$ or $\nvDash \Gamma, B \supset \Delta$.
6. If $\nvDash \Gamma \supset \Delta, A \vee B$ then $\nvDash \Gamma \supset \Delta, A, B$.

Proof. Analogous to the proof of the previous theorem. □

Theorem 1.28 (completeness)
If a CPRC sequent is not valid, then it is refutable in CPRC.

Proof. Suppose $\Gamma \supset \Delta$ is not valid and let M be a countermodel of it. We show by induction on $\text{len}(\Gamma \supset \Delta)$ that is it refutable in CPRC.

- $\text{len}(\Gamma \supset \Delta) = |\Gamma| + |\Delta|$.

  Then Γ and Δ contain only atomic formulas. Since $M \vDash \Gamma$ and for all $p \in \Delta$, $M \nvDash p$, the multisets Γ and Δ are disjoint, furthermore $\top \notin \Delta$. Thus $\Gamma \supset \Delta$ is an anti-axiom of CPRC.

- $\text{len}(\Gamma \supset \Delta) > |\Gamma| + |\Delta|$.

  Then there exists a formula $A \in \Gamma \cup \Delta$ with $\text{len}(A) > 1$.
  We distinguish by the form and position of A:

  - $A \equiv \neg p \qquad \Gamma \supset \Delta \equiv \Gamma', \neg p \supset \Delta$.

    Then $\Gamma' \supset \Delta, p$ is not valid (Lemma 1.27) and thus refutable by induction hypothesis. Using $(\neg\ \not\supset)$ we can deduce $\Gamma \supset \Delta$ from $\Gamma' \supset \Delta, p$.

  - $A \equiv B \vee D \qquad \Gamma \supset \Delta \equiv \Gamma', B \vee C \supset \Delta$.

    Then either $\Gamma', B \supset \Delta$ or $\Gamma', C \supset \Delta$ is not valid (Lemma 1.27) and thus refutable by induction hypothesis. We can then deduce $\Gamma \supset \Delta$ either from $\Gamma, B \supset \Delta$ with $(\cdot \vee\ \not\supset)$ or from $\Gamma, C \supset \Delta$ with $(\vee \cdot\ \not\supset)$.

  The other cases are analogous.

□

Macro-Rules

As in CPC we define macro-rules for the connectives $\rightarrow$ and $\leftrightarrow$ for more compact refutation representations. There is no implicit simplifications in any of those rules.

$$\frac{\Gamma \supset \Delta, A}{\Gamma, \mathbf{A} \rightarrow \mathbf{B} \supset \Delta}(\cdot\rightarrow\not\supset) \qquad \frac{\Gamma, B \supset \Delta}{\Gamma, \mathbf{A} \rightarrow \mathbf{B} \supset \Delta}(\rightarrow\cdot\not\supset) \qquad \frac{\Gamma, A \supset \Delta, B}{\Gamma \supset \Delta, \mathbf{A} \rightarrow \mathbf{B}}(\not\supset\rightarrow)$$

$$\frac{\Gamma, A, B \supset \Delta}{\Gamma, \mathbf{A} \leftrightarrow \mathbf{B} \supset \Delta}(\overset{\text{ll}}{\leftrightarrow}\not\supset) \qquad \frac{\Gamma, B \supset \Delta, A}{\Gamma \supset \Delta, \mathbf{A} \leftrightarrow \mathbf{B}}(\not\supset\overset{\text{rl}}{\leftrightarrow})$$

$$\frac{\Gamma \supset \Delta, A, B}{\Gamma, \mathbf{A} \leftrightarrow \mathbf{B} \supset \Delta}(\overset{\text{rr}}{\leftrightarrow}\not\supset) \qquad \frac{\Gamma, A \supset \Delta, B}{\Gamma \supset \Delta, \mathbf{A} \leftrightarrow \mathbf{B}}(\not\supset\overset{\text{lr}}{\leftrightarrow})$$

Again, these rules are but macros built from CPRC rules:

$$\frac{\dfrac{\Gamma \supset \Delta, A}{\Gamma, \neg\mathbf{A} \supset \Delta}(\neg\not\supset)}{\Gamma, \underbrace{\neg\mathbf{A} \vee \mathbf{B}}_{A\rightarrow B} \supset \Delta}(\cdot\vee\not\supset) \qquad \frac{\Gamma, B \supset \Delta}{\Gamma, \underbrace{\neg\mathbf{A} \vee \mathbf{B}}_{A\rightarrow B} \supset \Delta}(\vee\cdot\not\supset) \qquad \frac{\dfrac{\Gamma, A \supset \Delta, B}{\Gamma \supset \Delta, \neg\mathbf{A}, B}(\not\supset\neg)}{\Gamma \supset \Delta, \underbrace{\neg\mathbf{A} \vee \mathbf{B}}_{A\rightarrow B}}(\not\supset\vee)$$

$$\frac{\dfrac{\dfrac{\Gamma, A, B \supset \Delta}{\Gamma, \mathbf{A} \rightarrow \mathbf{B}, A \supset \Delta}(\rightarrow\cdot\not\supset)}{\Gamma, A \rightarrow B, \mathbf{B} \rightarrow \mathbf{A} \supset \Delta}(\rightarrow\cdot\not\supset)}{\Gamma, \underbrace{(\mathbf{A} \rightarrow \mathbf{B}) \wedge (\mathbf{B} \rightarrow \mathbf{A})}_{A\leftrightarrow B} \supset \Delta}(\wedge\not\supset) \qquad \frac{\dfrac{\Gamma, B \supset \Delta, A}{\Gamma \supset \Delta, \mathbf{B} \rightarrow \mathbf{A}}(\not\supset\rightarrow)}{\Gamma \supset \Delta, \underbrace{(\mathbf{A} \rightarrow \mathbf{B}) \wedge (\mathbf{B} \rightarrow \mathbf{A})}_{A\leftrightarrow B}}(\not\supset\wedge\cdot)$$

$$\frac{\dfrac{\dfrac{\Gamma \supset \Delta, A, B}{\Gamma, \mathbf{A} \rightarrow \mathbf{B} \supset \Delta, B}(\cdot\rightarrow\not\supset)}{\Gamma, A \rightarrow B, B \rightarrow A \supset \Delta}(\cdot\rightarrow\not\supset)}{\Gamma, \underbrace{(\mathbf{A} \rightarrow \mathbf{B}) \wedge (\mathbf{B} \rightarrow \mathbf{A})}_{A\leftrightarrow B} \supset \Delta}(\wedge\not\supset) \qquad \frac{\dfrac{\Gamma, A \supset \Delta, B}{\Gamma \supset \Delta, \mathbf{A} \rightarrow \mathbf{B}}(\not\supset\rightarrow)}{\Gamma \supset \Delta, \underbrace{(\mathbf{A} \rightarrow \mathbf{B}) \wedge (\mathbf{B} \rightarrow \mathbf{A})}_{A\leftrightarrow B}}(\not\supset\cdot\wedge)$$

As in the case of the calculus CPC, adding the macro to the calculus CPRC does not affect the result about soundness and completeness, since the rules only reflect applications of several core rules.

1.5.2 Examples

We give two simple examples to get a better understanding of the refutation calculus.

In the following example we look for a refutation of $p_1 \vee p_2, p_3 \to p_4 \supset q_1 \to q_2$. Since the antecedent and the succedent have no variables in common, it is clear that the succedent is not a logical consequence of the antecedent, hence our sequent is not valid.
We have four different classes of countermodels of our sequent:

$$\{M : M \vDash p_1, q_1, M \nvDash p_3, q_2\} \qquad \{M : M \vDash p_1, p_4, q_1, M \nvDash q_2\}$$
$$\{M : M \vDash p_2, q_1, M \nvDash p_3, q_2\} \qquad \{M : M \vDash p_2, p_4, q_1, M \nvDash q_2\}$$

Below there are four possible refutations of $p_1 \vee p_2, p_3 \to p_4 \supset q_1 \to q_2$. Each of it has an axiom that reflects one of the classes of models given above. There are other refutations of this sequent, but they will all use one of the axioms given below and vary only in the order of how the different rules are applied.

$$\dfrac{\dfrac{\dfrac{p_1, q_1 \supset p_3, q_2}{\mathbf{p_1} \vee \mathbf{p_2}, q_1 \supset p_3, q_2}(\cdot\vee\not\supset)}{p_1 \vee p_2, \mathbf{p_3} \to \mathbf{p_4}, q_1 \supset q_2}(\cdot\to\not\supset)}{p_1 \vee p_2, p_3 \to p_4 \supset \mathbf{q_1} \to \mathbf{q_2}}(\not\supset\to) \qquad \dfrac{\dfrac{\dfrac{p_1, p_4, q_1 \supset q_2}{\mathbf{p_1} \vee \mathbf{p_2}, p_4, q_1 \supset q_2}(\cdot\vee\not\supset)}{p_1 \vee p_2, \mathbf{p_3} \to \mathbf{p_4}, q_1 \supset q_2}(\to\cdot\not\supset)}{p_1 \vee p_2, p_3 \to p_4 \supset \mathbf{q_1} \to \mathbf{q_2}}(\not\supset\to)$$

$$\dfrac{\dfrac{\dfrac{p_2, q_1 \supset p_3, q_2}{\mathbf{p_1} \vee \mathbf{p_2} \supset p_3, q_2}(\vee\cdot\not\supset)}{p_1 \vee p_2, \mathbf{p_3} \to \mathbf{p_4}, q_1 \supset q_2}(\cdot\to\not\supset)}{p_1 \vee p_2, \mathbf{p_3} \to \mathbf{p_4} \supset q_1 \to q_2}(\not\supset\to) \qquad \dfrac{\dfrac{\dfrac{p_2, p_4, q_1 \supset q_2}{\mathbf{p_1} \vee \mathbf{p_2}, p_4, q_1 \supset q_2}(\vee\cdot\not\supset)}{p_1 \vee p_2, \mathbf{p_3} \to \mathbf{p_4}, q_1 \supset q_2}(\to\cdot\not\supset)}{p_1 \vee p_2, \mathbf{p_3} \to \mathbf{p_4} \supset q_1 \to q_2}(\not\supset\to)$$

In the second example we give the refutation of the sequent $p_1 \vee p_2, p_2 \leftrightarrow p_3, p_3 \to \neg p_4 \supset \neg p_4$. Supposing that $\mathcal{V} := \{p_1, p_2, p_3, p_4\}$, we have four possible models of the antecedent:

$$M_1 := \{p_1, p_2, p_3\} \qquad M_2 := \{p_1, p_4\} \qquad M_3 := \{p_1\} \qquad M_4 := \{p_2, p_3\}$$

The model M_2 is the only countermodel of our sequent. The proof (having an axiom reflecting M_2) is given below.
This example also points out the need of multisets in the sequent. Without the multiple occurrence of p_3 in the succedent of the axiom, we would not be able to refute the sequent in CPRC.

$$\dfrac{\dfrac{\dfrac{\dfrac{p_1, p_4 \supset p_2, p_3, p_3}{p_1 \supset p_2, p_3, p_3, \neg\mathbf{p_4}}(\not\supset\neg)}{\mathbf{p_1} \vee \mathbf{p_2} \supset p_2, p_3, p_3, \neg p_4}(\cdot\vee\not\supset)}{p_1 \vee p_2, \mathbf{p_2} \leftrightarrow \mathbf{p_3} \supset p_3, \neg p_4}(\overset{\text{rr}}{\leftrightarrow}\not\supset)}{p_1 \vee p_2, p_2 \leftrightarrow p_3, \mathbf{p_3} \to \neg\mathbf{p_4} \supset \neg p_4}(\cdot\to\not\supset)$$

Chapter 2

Proof Search in Classical Logic

In this chapter we discuss automatic proving in classical propositional logic. The chapter has three main sections. In the first section introduce a naive and inefficient proof search algorithm. Afterward we show two optimization techniques we use to improve our algorithm. In the third section we introduce a calculus which is especially tailored for proof search and that reflects one of the optimization techniques.

2.1 A Proof Search Algorithm

Besides being invertible, the deduction rules of CPC also enjoy the so called strong subformula property. These two properties allow us to write down a simple but naive proof search algorithm that needs no backtracking and which is known to terminate.

2.1.1 Some Rule Properties

Invertible Rules

When searching for a proof of a given sequent, it is desirable to only have rules that are invertible. If a rule is not invertible, all possibilities to apply the rule backwards must be taken into consideration to find the premises of the rule that lead to a given conclusion. Take for example the following pair of rules

$$\frac{\Gamma \supset \Delta, A}{\Gamma \supset \Delta, A \vee B}(\supset\cdot\vee) \qquad \frac{\Gamma \supset \Delta, B}{\Gamma \supset \Delta, A \vee B}(\supset\vee\cdot)$$

which are obviously not invertible. Using backward rule application for proof search would in this case result in an algorithm which needs backtracking for principal formulas of the form $A \vee B$ in the succedent.

As we have shown in the previous chapter, the rules of CPC are all invertible.

Subformula Property

For proof search another advantageous property of a deduction rule is the so called strong subformula property. If all rules of a calculus enjoy this property, a proof search algorithm relying on backward rule application is known to terminate.

Definition 2.1 (subformulas of a sequent, strong subformula property)
We define the multiset of *subformulas* of a CPC sequent $\Gamma \supset \Delta$ as follows:

$$\mathrm{sub}(\Gamma \supset \Delta) := \bigcup_{A \in \Gamma \cup \Delta} \mathrm{sub}(A).$$

A CPC rule $\dfrac{\mathrm{S}_1 \ldots \mathrm{S}_n}{\mathrm{S}}(\mathrm{R})$ is said to have the *strong subformula property* if

$$\mathrm{sub}(\mathrm{S}_i) \subsetneq \mathrm{sub}(\mathrm{S}) \text{ for all i with } 1 \leq i \leq n.$$

Since $\mathrm{sub}(A)$ is defined to be a multiset, the union operator in the definition above is to be understand as a multiset-union. We point this out because this is crucial for sequents made of multisets. Take for example the rule instance

$$\frac{p \supset p \vee q, p, q}{p \supset p \vee q, p \vee q}(\vee\supset).$$

Then we have $\mathrm{sub}(p \supset p \vee q, p, q) = \{p, p \vee q, p, q, p, q\}$ and $\mathrm{sub}(p \supset p \vee q, p \vee q) = \{p, p \vee q, p, q, p \vee q, p, q\}$ and thus this rule enjoys the strong subformula property. If we had used normal set union in the previous definition, both sets would be equal and thus this rule would not enjoy the strong subformula property.

2.1.2 A Naive and Inefficient Proof Search Algorithm

Since the rules of CPC are all invertible we obtain a correct provability algorithm by taking the sequent that is to be proved and then applying the rules backwards until an axiom is found or no more rule can be applied backwards. Termination is guaranteed because each rule of CPC enjoys the

$\textsc{cpcProvable}(p \wedge q \to r, \mathbf{p} \to \mathbf{q} \supset \neg p \vee r)$
1. $\textsc{cpcProvable}(\mathbf{p} \wedge \mathbf{q} \to \mathbf{r} \supset \neg p \vee r, p)$
1.1. $\textsc{cpcProvable}(\supset \neg p \vee r, p, \mathbf{p} \wedge \mathbf{q})$
1.1.1. $\textsc{cpcProvable}(\supset \neg \mathbf{p} \vee \mathbf{r}, p, p)$
$\textsc{cpcProvable}(\supset \neg \mathbf{p}, r, p, p)$
$\textsc{cpcProvable}(\mathbf{p} \supset r, p, \mathbf{p})$
1.1.2. $\textsc{cpcProvable}(\supset \neg \mathbf{p} \vee \mathbf{r}, p, q)$
$\textsc{cpcProvable}(\supset \neg \mathbf{p}, r, p, q)$
$\textsc{cpcProvable}(\mathbf{p} \supset r, \mathbf{p}, q)$
1.2. $\textsc{cpcProvable}(r \supset \neg \mathbf{p} \vee \mathbf{r}, p)$
$\textsc{cpcProvable}(\mathbf{r} \supset \neg p, \mathbf{r}, p)$
2. $\textsc{cpcProvable}(\mathbf{p} \wedge \mathbf{q} \to \mathbf{r}, q \supset \neg p \vee r)$
2.1. $\textsc{cpcProvable}(q \supset \neg p \vee r, \mathbf{p} \wedge \mathbf{q})$
2.1.1. $\textsc{cpcProvable}(q \supset \neg \mathbf{p} \vee \mathbf{r}, p)$
$\textsc{cpcProvable}(r, q \supset \neg \mathbf{p}, r, p)$
$\textsc{cpcProvable}(r, q, \mathbf{p} \supset r, \mathbf{p})$
2.1.2. $\textsc{cpcProvable}(r, \mathbf{q} \supset \neg p \vee r, \mathbf{q})$
2.2. $\textsc{cpcProvable}(r, q \supset \neg \mathbf{p} \vee \mathbf{r})$
$\textsc{cpcProvable}(\mathbf{r}, q \supset \neg p, \mathbf{r})$

Figure 2.1: Example proof search run

strong subformula property and thus with each backward application of a rule the complexity of the sequents that need to be proved is reduced. The pseudo code of this simple algorithm is given in Algorithm 1. We have chosen to also backward-apply the macro-rules.

Figure 2.1 shows a possible run of the proof search of the sequent $p \wedge q \to r, p \to q \supset \neg p \vee r$. The formulas printed in bold are the principal formulas that have been chosen arbitrarily by the algorithm. The indentation represents the recursion depth of the algorithm. If there are two recursive calls — i.e. for branching rules — then they are enumerated accordingly.

2.2 Optimization Techniques

In this section we present two optimization techniques that are used to improve the naive proof search algorithm. We first analyze a simple example to show that certain rules are to be prioritized. As a consequence the first improvement of our algorithm is then to process non-branching before branching rules. We have named this technique the classification of formulas. Afterward we have a detailed look at branching rules and will see that with a heuristic technique called use-check we are able to omit proving

Algorithm 1 Simple proof search in classical logic

```
1:  function cpcProvable(Γ ⊃ Δ)
2:      if Γ ⊃ Δ is an axiom then
3:          result := true
4:      else if Γ, Δ is atomic then
5:          result := false
6:      else if Γ is not atomic then
7:          choose A ∈ Γ which is not atomic
8:          Γ' := Γ \ {A}
9:          if A = ¬B then
10:             result := cpcProvable(Γ' ⊃ Δ, B)
11:         else if A = B ∨ C then
12:             result := cpcProvable(Γ', B ⊃ Δ) and
13:                       cpcProvable(Γ', C ⊃ Δ)
14:         else if A = B ∧ C then
15:             result := cpcProvable(Γ', B, C ⊃ Δ)
16:         else if A = B → C then
17:             result := cpcProvable(Γ' ⊃ Δ, B) and
18:                       cpcProvable(Γ', C ⊃ Δ)
19:         else if A = B ↔ C then
20:             result := cpcProvable(Γ', B, C ⊃ Δ) and
21:                       cpcProvable(Γ' ⊃ Δ, B, C)
22:     else
23:         choose A ∈ Δ which is not atomic
24:         Δ' := Δ \ {A}
25:         if A = ¬B then
26:             result := cpcProvable(Γ, B ⊃ Δ')
27:         else if A = B ∨ C then
28:             result := cpcProvable(Γ ⊃ Δ', B, C)
29:         else if A = B ∧ C then
30:             result := cpcProvable(Γ ⊃ Δ', B) and
31:                       cpcProvable(Γ ⊃ Δ', C)
32:         else if A = B → C then
33:             result := cpcProvable(Γ, B ⊃ Δ', C)
34:         else if A = B ↔ C then
35:             result := cpcProvable(Γ, B ⊃ Δ', C) and
36:                       cpcProvable(Γ, C ⊃ Δ', B)
37:     return result
```

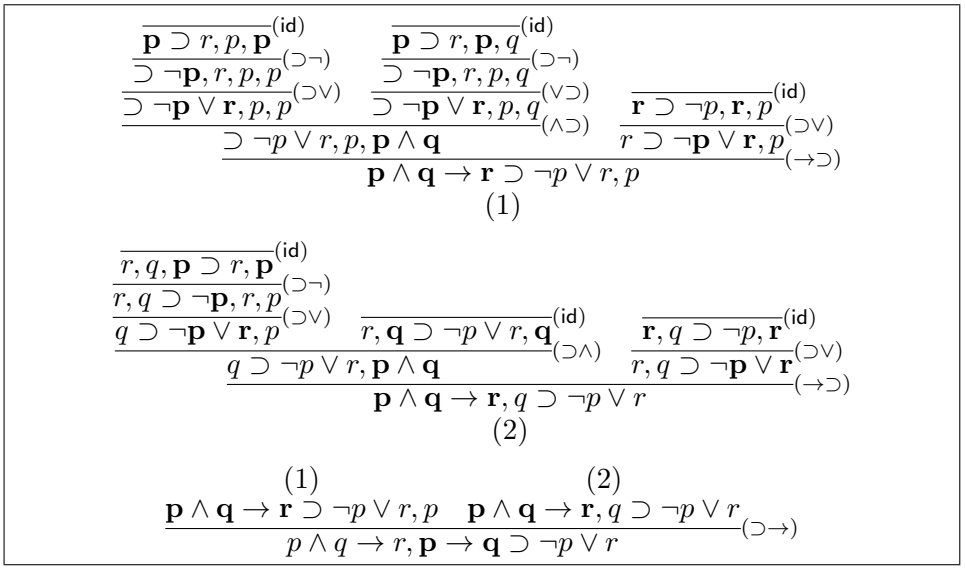


Figure 2.2: Proof to example run of Figure 2.1

certain branches.

2.2.1 Formula Classification

Let us look at the example run of Figure 2.1. The corresponding proof is given in Figure 2.2. A close examination of this simple example reveals that the formula $\neg p \vee r$ appears five times as principal formula. This is because some branching rules are applied backwards before $\neg p \vee r$ is processed. However, since $\neg p \vee r$ does not require a branching rule to be deduced and $\neg p \vee r$ is already a member of the antecedent of the sequent that is to be proved, it is better to first process $\neg p \vee r$ and its subformulas and then continue with those formulas that require branching rules to be deduced. The resulting run is shown in Figure 2.3, the corresponding proof is given in Figure 2.4. Comparing the two proofs shows that the new proof contains only one rule having $\neg p \vee r$ as principal formula. As a further consequence of processing $\neg p \vee r$ first, p is already part of the antecedent when the first branching rule has to be applied backwards and thus the left branch of the proof is reduced to an identity axiom.

The Idea of Formula Classification

The example above illustrates a general simple technique to improve the algorithm.

$\textsc{cpcProvable}(p \wedge q \to r, p \to q \supset \neg\mathbf{p} \vee \mathbf{r})$
$\textsc{cpcProvable}(p \wedge q \to r, p \to q \supset \neg\mathbf{p}, r)$
$\textsc{cpcProvable}(p \wedge q \to r, \mathbf{p} \to \mathbf{q}, p \supset r)$
1. $\textsc{cpcProvable}(p \wedge q \to r, \mathbf{p} \supset r, \mathbf{p})$
2. $\textsc{cpcProvable}(\mathbf{p} \wedge \mathbf{q} \to \mathbf{r}, q, p \supset r)$
2.1. $\textsc{cpcProvable}(q, p \supset r, \mathbf{p} \wedge \mathbf{q})$
2.1.1. $\textsc{cpcProvable}(q, \mathbf{p} \supset r, \mathbf{p})$
2.1.1. $\textsc{cpcProvable}(\mathbf{q}, p \supset r, \mathbf{q})$
2.2. $\textsc{cpcProvable}(\mathbf{r}, q, p \supset \mathbf{r})$

Figure 2.3: Improved example proof search run

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \wedge q \to r, \mathbf{p} \supset r, \mathbf{p}}(\mathsf{id}) \quad \dfrac{\dfrac{\dfrac{}{q, \mathbf{p} \supset r, \mathbf{p}}(\mathsf{id}) \quad \dfrac{}{\mathbf{q}, p \supset r, \mathbf{q}}(\mathsf{id})}{q, p \supset r, \mathbf{p} \wedge \mathbf{q}}(\supset\wedge) \quad \dfrac{}{\mathbf{r}, q, p \supset \mathbf{r}}(\mathsf{id})}{\mathbf{p} \wedge \mathbf{q} \to \mathbf{r}, q, p \supset r}(\to\supset)}{p \wedge q \to r, \mathbf{p} \to \mathbf{q}, p \supset r}(\to\supset)}{p \wedge q \to r, p \to q \supset \neg\mathbf{p}, r}(\supset\neg)}{p \wedge q \to r, p \to q \supset \neg\mathbf{p} \vee \mathbf{r}}(\supset\vee)$$

Figure 2.4: Proof corresponding to run of Figure 2.3

Formulas that are obtained by non-branching rules are to be processed before others.

We have named this technique *formula classification*. It is clear, that this heuristic technique will not generally result in shorter proofs. A corresponding example is shown in Figure 2.5. However, for those cases there are two remarks in favor of this technique. First, following this strategy will not introduce additional branches to the search tree. Second, in an implementation using heuristics as introduced later, the costs of applying a non-branching rule backwards are very small compared to the costs of applying branching rules backwards. This is because in contrast to non-branching rules, the state of the prover has to be saved before the premises of branching rules can be proved. As a consequence very little overhead is produced in cases where it is better to process a "branching" formula first.

2.2.2 Use-Check

Besides the obvious improvements there is another heuristic technique called use-check. With this technique we are able to omit whole search branches in certain circumstances. Based on an example we show how use-check works and have a look at the speedup it can lead to.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\mathbf{p}, p_1, p_2, p_3 \supset q, \mathbf{p}}^{(\mathsf{id})} \quad \overline{p, p_1, p_2, p_3, \mathbf{q} \supset \mathbf{q}}^{(\mathsf{id})}}{p, p_1, p_2, p_3, \mathbf{p} \to \mathbf{q} \supset q}(\to\supset)}{p, \mathbf{p_1} \wedge \mathbf{p_2}, p_3, p \to q \supset q}(\wedge\supset)}{p, \mathbf{p_1} \wedge \mathbf{p_2} \wedge \mathbf{p_3}, p \to q \supset q}(\wedge\supset)}$$

$$\dfrac{\overline{\mathbf{p}, p_1 \wedge p_2 \wedge p_3 \supset q, \mathbf{p}}^{(\mathsf{id})} \quad \overline{p, p_1 \wedge p_2 \wedge p_3, \mathbf{q} \supset \mathbf{q}}^{(\mathsf{id})}}{p, p_1 \wedge p_2 \wedge p_3, \mathbf{p} \to \mathbf{q} \supset q}(\to\supset)$$

Figure 2.5: Backward application of the branching rule leads in this example case into a shorter proof than the general strategy to apply non-branching rules first. The number of axioms in the proof stays the same

Motivation

With each backward application of a branching rule the prover potentially doubles the costs of finding a proof. Proof search can thus be accelerated when branchings are avoided. In some cases formula classification can avoid branchings, as the example above illustrates. However, those cases are limited to nodes where a backward rule application results in propositional variables. In larger proofs this normally happens very deep in the search tree. It is therefore obvious that only very small subproofs can be avoided with this technique.

The Idea of Use-Check

To accelerate proof search more efficiently it is desirable to avoid branchings very early in the search tree. A way to achieve this is the *use-check* technique. It allows us to omit proving the second premise of a branching rule. Use-Check is based on the fact that analyzing the proof of the first premise allows us in certain cases to conclude the provability of the second premise. The idea of the method is illustrated on the following example:
Suppose there is a backward rule application in the proof search of the form

$$\dfrac{\dfrac{\vdots}{\Gamma \supset \Delta, A} \quad \dfrac{\vdots}{\Gamma \supset \Delta, B}}{\Gamma \supset \Delta, \mathbf{A} \wedge \mathbf{B}}(\supset\wedge)$$

and the proof search procedure is successful in the left branch. If analyzing this proof reveals that already $\Gamma \supset \Delta$ is valid, then by monotonicity (Proposition 1.19) we know that $\Gamma \supset \Delta, B$ is valid and thus there is no need to search for a proof of it.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{p}, r, s \supset \mathbf{p}, q}(\mathsf{id})}{\mathbf{p} \wedge \mathbf{r}, s \supset p, q}(\wedge\supset) \quad \dfrac{}{\mathbf{q}, s \supset p, \mathbf{q}}(\mathsf{id})}{\mathbf{p} \wedge \mathbf{r} \vee \mathbf{q}, s \supset p, q}(\vee\supset) \quad \dfrac{\dfrac{\dfrac{}{\mathbf{p}, r, t \supset \mathbf{p}, q}(\mathsf{id})}{\mathbf{p} \wedge \mathbf{r}, t \supset p, q}(\wedge\supset) \quad \dfrac{}{\mathbf{q}, t \supset p, \mathbf{q}}(\mathsf{id})}{\mathbf{p} \wedge \mathbf{r} \vee \mathbf{q}, t \supset p, q}(\vee\supset)}{p \wedge r \vee q, \mathbf{s} \vee \mathbf{t} \supset p, q}$$

Figure 2.6: An example proof illustrating the use-check method

cpcProvable($p \wedge r \vee q, \mathbf{s} \vee \mathbf{t} \supset p, q$)
1. cpcProvable($\mathbf{p} \wedge \mathbf{r} \vee \mathbf{q}, s \supset p, q$)
1.1. cpcProvable($\mathbf{p} \wedge \mathbf{r}, s \supset p, q$)
cpcProvable($\mathbf{p}, r, s \supset \mathbf{p}, q$)
relevant: $p \supset p$
active formula p is relevant, mark principal formula as relevant
relevant: $p \wedge r \supset p$
active formula $p \wedge r$ is relevant, need to prove right branch
1.2. cpcProvable($\mathbf{q}, s \supset p, \mathbf{q}$)
relevant: $q \supset q$
active formula q is relevant, merge information about relevant formulas
relevant: $p \wedge r \vee q \supset p, q$
active formula s is not relevant, no need to prove second branch
relevant: $p \wedge r \vee q \supset p, q$
2. cpcProvable($\mathbf{p} \wedge \mathbf{r} \vee \mathbf{q}, t \supset p, q$) omitted (is provable)

Figure 2.7: An example proof search where use-check succeeds

Algorithmic Aspects

For an efficient algorithm it is important to combine the analysis of the proof with the proof search. We will explain on an example proof how this can be done and what kind of information we must store during proof search for the use-check method.

The proof in Figure 2.6 represents a possible run of the simple prover for the sequent $p \wedge r \vee q, s \vee t \supset p, q$.
In the subproof of the first left branch we see that the formula s is not used as principal formula in any axiom of the proof of $\mathbf{p} \wedge \mathbf{r} \vee \mathbf{q}, s \supset p, q$. We say that s is not relevant for proving this sequent. The use-check method is based on this relevance of formulas. The idea is to mark the relevant formulas in the axioms and to propagate this information towards the root of the search tree when returning from the recursive function call. The example above would result in the search shown in Figure 2.7.

Propagating Relevance Information

The way we propagate the information about relevant formulas along the search tree is straightforward. When encountering an axiom we mark its principal formulas as being relevant. Then this relevance is passed along every rule application of the proof that we have found:

- For non-branching rules we mark the principal formula as relevant if any of its active formula in the premise is marked as relevant.
- For branching rules we have the following cases:
  1. *In both branches an active formula in the premise is marked as relevant.*
     In that case we merge the information about relevant formulas and mark the principal formula as relevant.
  2. *In the left branch no active formula in the premise is marked as relevant.*
     In that case we can omit proving the right branch and take over the information about relevant formulas from the proof of the left premise.
  3. *In the left branch but not in the right branch an active formula in the premise is marked as relevant.*
     In that case we could have omitted proving the left branch. We therefore take over the information about relevant formulas from the proof of the right premise.

Remark 2.2 (dependency of formulas)
The propagation of relevance information represents the dependency of formulas in the proof. Clearly, for each rule R in a proof the active formulas in the premises of R depend directly on the principal formula of R. Dependency is then defined as the transitive extension of direct dependency.

The Problem of Multiple Formula Occurrence

In each sequent of the example proof above there are no multiple occurrences of the same formula on the antecedent or succedent. Unfortunately this is not generally the case. As a consequence the structure of a formula is not sufficient to decide its dependency, as the example in Figure 2.8 illustrates.

It is thus necessary to add additional information to our sequents in order to derive the formula-dependencies. This can for example be achieved by labeling each formula with a dependency information.
Since for our purpose only the branchings are of importance, it is sufficient for us to know of each formula from which branchings in the search tree

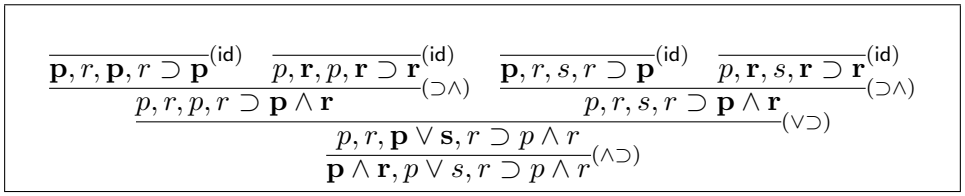


Figure 2.8: In this example proof there are multiple occurrences of formulas in the antecedent of several sequent. Since sequents are based on multisets, it is not possible to decide the dependency of certain formulas without further information.

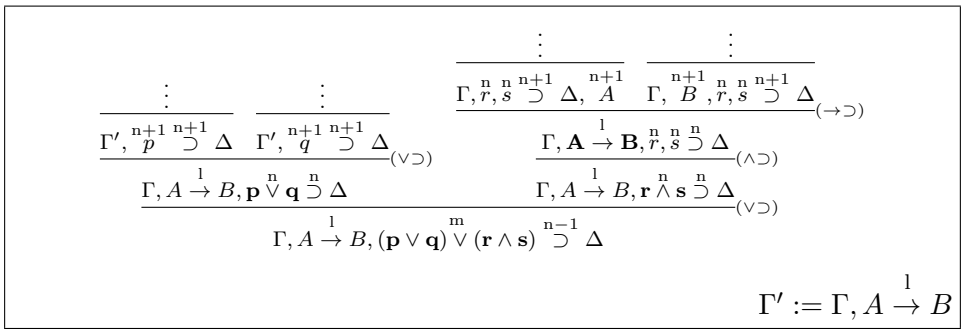


Figure 2.9: Example proof to illustrate formula labeling

it depends of. We therefore label each formula in a proof with a natural number that corresponds to the branching depth of the search tree in which it was created. We do this the following way:

- The initial branching depth is set to 0 and all formulas of the sequent that is to be proved are labeled with it.
- When a non-branching rule is processed, the active formulas in the premise inherit the label of the principal formula.
- When a branching rule is processed, the value assigned to the current branching depth is incremented and the active formulas in the premises are labeled with this new value.

The example proof in Figure 2.9 illustrates this proceeding. $p \to q$ and $r \wedge s$ are active formulas in the premise of a branching rule and thus labeled with the natural number assigned to the branching depth — denoted by the label on the sequent separator. r and s inherit the label of $r \wedge s$ since they result from the non-branching rule $(\wedge \supset)$. A and B then are again labeled with the number assigned to the corresponding branching depth.

For the moment we use labeled formulas in sequents informally but will give a correct definition in a later section where we introduce the calculus CPC2.

Using Sets to Store Dependencies

Having labeled formulas it is no longer necessary to mark the formulas as being used. Instead we keep with each sequent $\Gamma \supset \Delta$ a set $\mathcal{D}$ holding the labels of those formulas that are relevant in the proof found for $\Gamma \supset \Delta$. Each sequent in our proof is thus of the form $\mathcal{D}; \Gamma \overset{\text{n}}{\supset} \Delta$, where n denotes the value of the current branching depth.

With this kind of sequents, the information about relevant formulas is propagated in an easier way:

- For a non branching rule $\dfrac{\mathcal{D}; \Gamma' \overset{\text{n}}{\supset} \Delta'}{\mathcal{D}; \Gamma \overset{\text{n}}{\supset} \Delta}$(R), $\mathcal{D}$ is left unchanged.
- For a branching rule $\dfrac{\mathcal{D}_1; \Gamma_1 \overset{\text{n}}{\supset} \Delta_1 \quad \mathcal{D}_2; \Gamma_2 \overset{\text{n}}{\supset} \Delta_2}{\mathcal{D}; \Gamma \overset{\text{n-1}}{\supset} \Delta}$(R) whose principal formula carries the label m, we have the following cases:
  1. $\text{n} \in \mathcal{D}_1$ and $\text{n} \in \mathcal{D}_2$ and $\mathcal{D} = (\mathcal{D}_1 \cup \mathcal{D}_2 \cup \{\text{m}\}) \setminus \{\text{n}\}$.
     In that case in the proofs of both premises at least one active formula in the premise is relevant, i.e. the branching is necessary. $\mathcal{D}$ must thus contain all labels in $\mathcal{D}_1$ and $\mathcal{D}_2$ and the label of the principal formula m. The information about the branching depth of the premises can be dropped since it is of no interest anymore and according to our procedure of labeling the formulas, there are no other formulas in the conclusion carrying this label.
  2. $\text{n} \notin \mathcal{D}_1$ and $\mathcal{D} = \mathcal{D}_1$.
     In that case, no active formula in the premise is relevant in the proof of the first premise. We take over the relevance information of the first premise and omit proving the second premise.
  3. $\text{n} \in \mathcal{D}_1$ and $\text{n} \notin \mathcal{D}_2$ and $\mathcal{D} = \mathcal{D}_2$.
     In that case no active formula in the premise is relevant in the proof of the second premise. As in the second case we take over the relevance information of the second premise

The Importance of the Axioms for Use-Check

When encountering an axiom in the proof search procedure, there will be generally more than one possibility to choose the principal formulas. Since propagating the information about relevant formulas is based on the axioms, the choice of the principal formulas in the axioms fully determines at which branch in the proof search use-check will succeed.

Suppose we have the following situation:

$$\dfrac{\dfrac{}{\Gamma, \overset{\mathrm{n}}{p} \overset{\mathrm{n}}{\supset} \Delta, \overset{\mathrm{k_1}}{p}, \ldots, \overset{\mathrm{k_l}}{p}}(\mathsf{id}) \quad \dfrac{\vdots}{\Gamma, \overset{\mathrm{n}}{q} \overset{\mathrm{n}}{\supset} \Delta, \overset{\mathrm{k_1}}{p}, \ldots, \overset{\mathrm{k_l}}{p}}}{\dfrac{\Gamma, \mathbf{p} \overset{\mathrm{m}}{\vee} \mathbf{q} \overset{\mathrm{n-1}}{\supset} \Delta, \overset{\mathrm{k_1}}{p}, \ldots, \overset{\mathrm{k_l}}{p}}{\vdots}}(\vee\supset)$$

This leaves us to chose $\overset{\mathrm{n}}{p}$ together with any element of $\left\{\overset{\mathrm{k_1}}{p}, \ldots, \overset{\mathrm{k_l}}{p}\right\}$.

In such a case, there are several simple strategies of how to choose.

1. *Choose the p that depends on the least branching nodes.*

   With this strategy we concentrate on fewer branchings and therefore raise the probability that a use-check will succeed.

2. *Choose the p with the smallest label.*

   A p with a small label is created first in the search tree. Thus this p will probably appear in more axioms of the proof than the other p's do. Chances that the same p is also used as a principal formula in other axioms are therefore high.

   For any other p there is a certain probability that it depends on the same node as a p with a bigger label does. By choosing the one with the smallest label this will never be the case.

3. *Choose the p with the largest label.*

   With this strategy we concentrate on the branching nodes that occur late in the search tree. Like that we increase the chances that use-check succeeds in a branching node early in the search tree.

A Proof Search Algorithm With Use-Check

We close this section with an algorithm that does use-check. The pseudocode is given in Algorithm 2 and consists of three function whose arguments are all passed by value. For multiset of labeled formulas we here use the symbols $\overline{\Gamma}$ and $\overline{\Delta}$.

cpcProvableUC This is the interface function. Its arguments are two multisets of formulas representing the antecedent and succedent. At first the formulas in the given sequent are all labeled with 0. Then classify is called to start the backward proof search. This call will then either return $\emptyset$ or $\{0\}$ which is equal to non-validity and validity of the sequent.

Algorithm 2 Proof search with use-check in classical logic

```
1:  function cpcProvableUC(Γ, Δ)
2:      Γ̄ := {⁰A : A ∈ Γ}; Δ̄ := {⁰A : A ∈ Δ}
3:      D := classify(Γ̄, Δ̄, ∅, ∅, ∅, ∅)
4:      return D = ∅ ? false : true
5:
6:  function classify( Γ̄_c, Δ̄_c, Γ̄_b, Δ̄_b, Γ̄_a, Δ̄_a)
7:      if Γ̄_c ∪ Δ̄_c ≠ ∅ then
8:          if Γ̄_c ≠ ∅ then
9:              choose ⁿA ∈ Γ̄_c; Γ̄_c := Γ̄_c \ {ⁿA}
10:             if ⁿA = ¬ⁿB then  Δ̄_c := Δ̄_c ∪ {ⁿB}
11:             else if ⁿA = ⁿB ∧ ⁿC then  Γ̄_c := Γ̄_c ∪ {ⁿB, ⁿC}
12:             else if ⁿA is atomic then  Γ̄_a := Γ̄_a ∪ {ⁿA}
13:             else Γ̄_b := Γ̄_b ∪ {ⁿA}                    ▷ ⁿA = ⁿB ∨ ⁿC
14:         else proceed analogously with Δ̄_c
15:         if Γ̄_a ⊃ Δ̄_a is an axiom then
16:             select principal formulas of axiom
17:             D := {n : ⁿA is principal formula of axiom}
18:         else
19:             D := classify(Γ̄_c, Δ̄_c, Γ̄_b, Δ̄_b, Γ̄_a, Δ̄_a)
20:     else if Γ̄_b ∪ Δ̄_b ≠ ∅ then
21:         D := branch(Γ̄_b, Δ̄_b, Γ̄_a, Δ̄_a)
22:     else D := ∅                          ▷ only atoms left but no axiom
23:     return D
24:
25: function branch(Γ̄_b, Δ̄_b, Γ̄_a, Δ̄_a)
26:     n := max(labels(Γ̄_b ∪ Δ̄_b ∪ Γ̄_a ∪ Δ̄_a)) + 1
27:     if Γ̄_b ≠ ∅ then
28:         choose ᵐA ∈ Γ̄_b; Γ̄_b := Γ̄_b \ ᵐA                ▷ ᵐA = ⁿB ∨ ᵐC
29:         D_1 := classify({ⁿB}, ∅, Γ̄_b, Δ̄_b, Γ̄_a, Δ̄_a)
30:         if n ∈ D_1 then                          ▷ use-check not successful
31:             D_2 := classify({ⁿC}, ∅, Γ̄_b, Δ̄_b, Γ̄_a, Δ̄_a)
32:             if n ∈ D_2 then                      ▷ use-check not successful
33:                 D := (D_1 ∪ D_2 ∪ {m}) \ {n}
34:             else D := D_2          ▷ use-check successful in right branch
35:         else D := D_1              ▷ use-check successful in left branch
36:     else proceed analogously with Δ̄_b
37:     return D
```

CLASSIFY This function does the classification of the formulas and checks for axioms. Its arguments are 6 multisets of labeled formulas representing three groups of formulas in the antecedent ($\overline{\Gamma}_x$) and succedent ($\overline{\Delta}_x$). $\overline{\Gamma}_c$ and $\overline{\Delta}_c$ hold the formulas that need yet to be classified, $\overline{\Gamma}_b$ and $\overline{\Delta}_b$ hold the formulas that require a branching rule to be deduced and $\overline{\Gamma}_a$ and $\overline{\Delta}_a$ hold the atomic formulas.
For lack of space we only give the pseudocode for the formulas in the antecedent. The processing of the formulas in the succedent is analogical.
The function has a first triple case distinction.

1. If there are still formulas to classify, an arbitrary chosen formula from $\overline{\Gamma}_c$ or $\overline{\Delta}_c$ is classified (lines 7–14). This means that depending on the form of the chosen formula we either apply the non-branching rule backwards (lines 10 and 11), put the formula into the corresponding set of atomic formulas (line 12) or into the corresponding set of formulas that require a branching rule (line 13).
   To keep the pseudocode simple we process formulas in the antecedent first (line 8). In an implementation a good strategy is to select the formula alternating from the antecedent and succedent.
   After classifying the chosen formula we check whether we have reached an axiom[1].
   If an axiom is reached we are free to select the principal formulas of our axiom. The set $\mathcal{D}$ holding the labels that we return is then initialized to the labels of those formulas that we have selected as principal formulas (lines 15–17).
   The selection of the principal formulas can be arbitrary or according to one of the previously given strategies. For the case where we would like to select the formula depending on the least branching nodes, further information needs to be passed to our function for a proper selection of the principal formulas. The other strategies (smallest/largest label) can be followed without additional information.
   If no axiom is reached we continue by recursively call CLASSIFY (lines 18 and 19).

2. If there are no more formulas to classify but formulas that require a branching rule, we continue by calling BRANCH (lines 20 and 21).

3. If neither formulas to classify nor formulas that require a branching rule are left, the prover failed and will set our result $\mathbb{D}$ to an

[1] Checking for an axiom is in fact only necessary if any formulas have been added to $\overline{\Gamma}_a$ or $\overline{\Delta}_a$, but for simplicity it is written this way

empty set. (line 22).

BRANCH This function applies the two branching rules ($\vee$ $\supset$) and ($\supset$ $\wedge$) backwards. Its arguments are the same as the last four arguments of the function CLASSIFY. A call to it is only done if there are still formulas left that must be deduced with a branching rule and no more formulas need to be classified.
For lack of space we only give the pseudocode for the formulas in the antecedent. The processing of the formulas in the succedent is analogical.
First we calculate the label n which the active formulas in the premise will carry (line 26). It is easy to see that according to our labeling this is the current branching depth of the search tree.
Then an arbitrary branching formula is chosen (line 27). As in CLASSIFY we process formulas in the antecedent first to keep the pseudocode simple. Also here, a good strategy in an implementation would be to select the formula alternating from the antecedent and succedent.
What now follows is according to the case distinction on page 31.
We first try to prove the left premise by calling classify (line 29). The set of labels $\mathcal{D}_1$ we get in return either contains n (the label of the active formula in the premise $\overset{\text{n}}{B}$) or not.

- $\text{n} \notin \mathcal{D}_1$: Then either the left premise was not provable or it was provable but no atomic formula that has origin in $\overset{\text{n}}{B}$ was used as principal formula in any of the axioms of the proof we have found. The second possibility of this case corresponds to case 2 on page 31. We thus simply return the use-set returned by CLASSIFY (line 35).

- $\text{n} \in \mathcal{D}_1$: Then the left premise was provable and an atomic formula that has origin in $\overset{\text{n}}{B}$ was used as principal formula in the proof that we have found for it. In this case we have to search for a proof of the right premise (line 31). According to the set $\mathcal{D}_2$ returned by CLASSIFY we then again have two cases depending on whether n is in $\mathcal{D}_2$ or not.

  - $\text{n} \notin \mathcal{D}_2$: Then either the premise was not provable or it was provable but no atomic formula that has origin in $\overset{\text{n}}{C}$ was used as principal formula in any of the axioms of the proof we have found. The second possibility of this case corresponds to case 3 on page 31. We thus simply return the use-set returned by CLASSIFY (line 34).

  - $\text{n} \in \mathcal{D}_2$: Then the right premise was provable and an atomic formula that has origin in $\overset{\text{n}}{C}$ was used as principal formula

in the proof that we have found for it. This corresponds to case 1 on page 31. We calculate the set of labels to return accordingly (line 33).

2.3 A Calculus With Use-Check

In this section we present a calculus for classical propositional logic which is tailored to our proof search algorithm. The calculus makes use of sequents that contain the heuristic information we need for the use-check method.

Besides further illustrating the method of use-check, the calculus is mainly presented to give a formal proof of the soundness of use-check. Later in this section we also show that the calculus can be used to compute minimal sequents, which are of interest for doing proof search in default logic.

2.3.1 The Calculus CPC2

In the previous section we have introduced labeled formulas informally. We now give the formal definitions of a labeled formula and a labeled sequent.

Definition 2.3 (labeled formula)
A *labeled formula* is a pair $\langle A, \mathrm{n}\rangle \in \mathcal{L} \times \mathbb{N}$ denoted by $\overset{\mathrm{n}}{A}$ where n is called the label of the formula.

We write $\mathcal{L}_{\mathbb{N}}$ to denote the set of all labeled formulas.

Definition 2.4 (labels($\overline{\Gamma}$)**)**
Let $\overline{\Gamma}$ be a (multi)set of labeled formulas. We define the set of labels of $\overline{\Gamma}$ as:

$$\mathrm{labels}(\overline{\Gamma}) := \left\{ \mathrm{n} : \overset{\mathrm{n}}{A} \in \overline{\Gamma} \text{ for some } A \right\}.$$

Since the calculus CPC2 encloses use-check, we need sequents that contain the heuristic information about which of the formulas is relevant in the proof. We do this, as in the previous section, by extending the sequent with a use-set that may only contain labels that occur in the labeled formulas of the sequent.

In the previous section we also used to have a label on the sequent itself. This label represented the current branching depth in the proof search algorithm and was mainly used for illustration. Since there is no need for such a label in the calculus CPC2, our definition of a labeled sequent does not contain it.

Definition 2.5 (labeled sequent)
A CPC2 *sequent* or *labeled sequent* is a triple $\langle \mathcal{D}, \overline{\Gamma}, \overline{\Delta}\rangle$ consisting of two multisets $\overline{\Gamma} \subseteq \mathcal{L}_{\mathbb{N}}$ and $\overline{\Delta} \subseteq \mathcal{L}_{\mathbb{N}}$ and the *use-set* $\mathcal{D} \subseteq \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$.

We write $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ to denote a labeled sequent and use the symbols $\overline{\Gamma}$ and $\overline{\Delta}$ for (multi-)sets of $\mathcal{L}_{\mathbb{N}}$-formulas.

Given the natural numbers $\mathrm{n}_1, \ldots \mathrm{n}_m$ and sets $\mathcal{D}_1 \subset \mathbb{N}, \mathcal{D}_2 \subset \mathbb{N}$ we normally write

$$\begin{array}{rl} \mathcal{D}_1, \mathcal{D}_2, \mathrm{n}_1, \ldots, \mathrm{n}_m; \overline{\Gamma} \supset \overline{\Delta} & \text{for} \quad \mathcal{D}_1 \cup \mathcal{D}_2 \cup \{\mathrm{n}_1, \ldots, \mathrm{n}_m\}; \overline{\Gamma} \supset \overline{\Delta} \quad \text{and} \\ \mathrm{n}_1, \ldots, \mathrm{n}_m; \overline{\Gamma} \supset \overline{\Delta} & \text{for} \quad \{\mathrm{n}_1, \ldots, \mathrm{n}_m\}; \overline{\Gamma} \supset \overline{\Delta}. \end{array}$$

The use-set in a labeled sequent holds the labels of those formulas that are relevant for the proof. We use it to transform the labeled sequent into a CPC sequent such that formulas that carry labels which are in the use-set will occur in the transformed sequent. Validity of a labeled sequent is then defined as validity of the corresponding CPC sequent.

Definition 2.6 ($\overline{\Gamma}\downarrow_{\mathcal{D}}$, $\overset{\mathrm{n}}{\Gamma}$)
Given a (multi)-set of labeled formulas $\overline{\Gamma} \subset \mathcal{L}_{\mathbb{N}}$ and a set $\mathcal{D} \subset \mathbb{N}$ we write $\overline{\Gamma}\downarrow_{\mathcal{D}}$ for the (multi)-set $\left\{ A : \text{ there exists } \mathrm{n} \in \mathcal{D} \text{ such that } \overset{\mathrm{n}}{A} \in \overline{\Gamma} \right\}$.

Given a (multi)-set of formulas $\Gamma \subset \mathcal{L}$ and a natural number n we write $\overset{\mathrm{n}}{\Gamma}$ to denote the (multi)-set $\left\{ \overset{\mathrm{n}}{A} : A \in \Gamma \right\}$.

Definition 2.7 (length of a labeled sequent)
The length of a labeled sequent is defined to be the sum of the length of its formulas.

$$\mathrm{len}(\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}) := \mathrm{len}(\overline{\Gamma}\downarrow_{\mathbb{N}} \supset \overline{\Delta}\downarrow_{\mathbb{N}})$$

Definition 2.8 (valid labeled sequent)
A labeled sequent $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is defined to be valid if $\overline{\Gamma}\downarrow_{\mathcal{D}} \supset \overline{\Delta}\downarrow_{\mathcal{D}}$ is valid. We denote this as usual with

$$\models \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}.$$

It is important to see, that $\mathcal{D}$ is relevant for a labeled sequent to be valid. We have for example

$$\models 1, 2; \supset \overset{1}{A}, \neg \overset{2}{A} \qquad \text{whereas} \qquad \not\models 1; \supset \overset{1}{A}, \neg \overset{2}{A}.$$

Definition 2.9 (the calculus CPC2)
We define the classical sequent calculus CPC2 to have the deduction rules as given in Figure 2.10.

The main difference between CPC2 and CPC is that for rules with two premises there are two additional rules with just one premise but with

$$\frac{}{\mathcal{D}, \mathrm{n}, \mathrm{m}; \overline{\Gamma}, \overset{\mathrm{n}}{\mathbf{p}} \supset \overline{\Delta}, \overset{\mathrm{m}}{\mathbf{p}}}(\mathsf{id}) \qquad \frac{}{\mathcal{D}, \mathrm{n}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\top}}(\top)$$

$$\frac{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}}{\mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{\neg \mathbf{A}} \supset \overline{\Delta}}(\neg\supset) \qquad \frac{\mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}}{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\neg \mathbf{A}}}(\supset\neg)$$

$$\frac{\mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}}{\mathcal{D}; \overline{\Gamma}, \mathbf{A} \overset{\mathrm{m}}{\vee} \mathbf{B} \supset \overline{\Delta}}(\cdot\vee\supset)^a \qquad \frac{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}, \overset{\mathrm{n}}{B}}{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \mathbf{A} \overset{\mathrm{n}}{\vee} \mathbf{B}}(\supset\vee)$$

$$\frac{\mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}}{\mathcal{D}; \overline{\Gamma}, \mathbf{A} \overset{\mathrm{m}}{\vee} \mathbf{B} \supset \overline{\Delta}}(\vee\cdot\supset)^a \qquad \frac{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}}{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \mathbf{A} \overset{\mathrm{m}}{\wedge} \mathbf{B}}(\supset\cdot\wedge)^a$$

$$\frac{\mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}}{\mathcal{D}; \overline{\Gamma}, \mathbf{A} \overset{\mathrm{n}}{\wedge} \mathbf{B} \supset \overline{\Delta}}(\wedge\supset) \qquad \frac{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{B}}{\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \mathbf{A} \overset{\mathrm{m}}{\wedge} \mathbf{B}}(\supset\wedge\cdot)^a$$

$$\frac{\mathcal{D}_1, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta} \quad \mathcal{D}_2, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}}{\mathcal{D}_1, \mathcal{D}_2, \mathrm{m}, \mathrm{n}^*; \overline{\Gamma}, \mathbf{A} \overset{\mathrm{m}}{\vee} \mathbf{B} \supset \overline{\Delta}}(\vee\supset)^b$$

$$\frac{\mathcal{D}_1, \mathrm{n}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A} \quad \mathcal{D}_2, \mathrm{n}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{B}}{\mathcal{D}_1, \mathcal{D}_2, \mathrm{m}, \mathrm{n}^*; \overline{\Gamma} \supset \overline{\Delta}, \mathbf{A} \overset{\mathrm{m}}{\wedge} \mathbf{B}}(\supset\wedge)^b$$

[a] $\mathrm{n} \notin \mathcal{D}$
[b] $\mathrm{n} \notin \mathcal{D}_1 \cup \mathcal{D}_2$
* if $\mathrm{n} \in \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$, otherwise without n

Figure 2.10: Deduction rules of CPC2

implicit weakening. These rules reflect exactly the use-check method as described on page 31. The rules $(\vee \supset), (\cdot\vee \supset)$ and $(\vee\cdot \supset)$ for example correspond to the cases 1,2 and 3 of how relevance information is propagated in proof search. In exactly these rules an arbitrary formula label is introduced for the principal formula. This corresponds to the procedure of labeling the active formulas in the premises of a branching rule with the value assigned to the current branching depth.

We have chosen to share some rule names between the calculi CPC and CPC2. However, when referring to a name of a rule it is normally clear from the context which calculus we refer to.

It is important that an arbitrary label is introduced for the principal formula in the rules $(\cdot\vee \supset), (\vee\cdot \supset), (\vee \supset)$. If the principal formula would inherit the label of the active formulas in the premises, unwanted dependencies would be introduces. Consider for example the following situation where the principal formula inherits the label of the active formulas in the premises.

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \mathcal{D}_1;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{A}, \overset{\mathrm{m}}{C} & \mathcal{D}_2;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{B}, \overset{\mathrm{m}}{C}\end{array}}{\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{A \wedge B}, \overset{\mathrm{m}}{C}}(\supset\wedge)}{\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{(A \wedge B)} \vee C}(\supset\vee)$$

Then $\mathrm{m} \in \mathcal{D}_1$, that is we have principal formulas in the axioms of the subproof of $\mathcal{D}_1;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{A}, \overset{\mathrm{m}}{C}$ that originate in formulas carrying the label m. We can not distinguish whether they originate in $\overset{\mathrm{m}}{A}$ or $\overset{\mathrm{m}}{C}$ and are thus not certain, whether $\overset{\mathrm{m}}{A}$ is really needed since it is possible that all of them originate in $\overset{\mathrm{m}}{C}$.
In the right premise we have a similar situation.

By allowing different labels for the active formulas in the premise and the principal formula we can get rid of this uncertainty.

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \mathcal{D}_1;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}, \overset{\mathrm{m}}{C} & \mathcal{D}_2;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{B}, \overset{\mathrm{m}}{C}\end{array}}{\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{A \wedge B}, \overset{\mathrm{m}}{C}}(\supset\wedge)}{\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{(A \wedge B)} \vee C}(\supset\vee)$$

Provided that we have the same structure in the subproof of $\mathcal{D}_1;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}, \overset{\mathrm{m}}{C}$ as in the subproof of $\mathcal{D}_1;\overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{A}, \overset{\mathrm{m}}{C}$, it is now possible that $\mathrm{m} \in \mathcal{D}_1$ but

$\mathrm{n} \notin \mathcal{D}_1$. Then instead of $(\supset \wedge)$ we would have to use $(\supset \cdot\wedge)$ to deduce $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, A \overset{\mathrm{m}}{\wedge} B, \overset{\mathrm{m}}{C}$ and can so omit the right premise.
In the right premise we a similar situation.

2.3.2 Soundness and Completeness

In this section we show that CPC2 is sound and complete. We start by showing the soundness of the rules. Then we give an auxiliary lemma that shows that certain rules of CPC2 are invertible if the labels of the principal and active formulas in the premise are equal. Based on that lemma we then show the completeness of CPC2.

Theorem 2.10 (soundness of CPC2)
If a labeled sequent $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is deducible in CPC2 then it is valid.

Proof. It is obvious that the axioms of CPC2 are sound. Since the other rules can be grouped into rules of similar form, we only show soundness for two rules and leave it up to the reader to verify the other rules.

- $\dfrac{\mathcal{D}_1, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta} \quad \mathcal{D}_2, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}}{\mathcal{D}_1, \mathcal{D}_2, \mathrm{m}, \mathrm{n}^*; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}} (\vee\supset)$ (cf. Figure 2.10 for requirements)

  Suppose $\vDash \mathcal{D}_1, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $\vDash \mathcal{D}_2, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$. From the definition of validity we then know that

  $$\vDash \overline{\Gamma}{\downarrow}_{\mathcal{D}_1 \cup \{\mathrm{n}\}}, A \supset \overline{\Delta}{\downarrow}_{\mathcal{D}_1 \cup \{\mathrm{n}\}} \qquad \vDash \overline{\Gamma}{\downarrow}_{\mathcal{D}_2 \cup \{\mathrm{n}\}}, B \supset \overline{\Delta}{\downarrow}_{\mathcal{D}_2 \cup \{\mathrm{n}\}}.$$

  We distinguish two cases:

  - $\mathrm{n} \in \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$

    Then let $\mathcal{D} := \mathcal{D}_1 \cup \mathcal{D}_2 \cup \{\mathrm{n}\}$. Since $\mathrm{n} \in \mathcal{D}$ and $\mathcal{D}_1$ and $\mathcal{D}_2$ are subsets of $\mathcal{D}$ we know $\overline{\Gamma}{\downarrow}_{\mathcal{D}_i \cup \{\mathrm{n}\}} \subseteq \overline{\Gamma}{\downarrow}_{\mathcal{D}}$ and $\overline{\Delta}{\downarrow}_{\mathcal{D}_i \cup \{\mathrm{n}\}} \subseteq \overline{\Delta}{\downarrow}_{\mathcal{D}}$ for $i \in \{1, 2\}$. By monotonicity we obtain

    $$\vDash \overline{\Gamma}{\downarrow}_{\mathcal{D}}, A \supset \overline{\Delta}{\downarrow}_{\mathcal{D}} \qquad \vDash \overline{\Gamma}{\downarrow}_{\mathcal{D}}, B \supset \overline{\Delta}{\downarrow}_{\mathcal{D}}.$$

    Hence $\vDash \overline{\Gamma}{\downarrow}_{\mathcal{D}}, A \vee B \supset \overline{\Delta}{\downarrow}_{\mathcal{D}}$ and thus $\mathcal{D}, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$, which is equivalent to $\mathcal{D}_1, \mathcal{D}_2, \mathrm{m}, \mathrm{n}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$, is valid.

  - $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$

    Then let $\mathcal{D} := \mathcal{D}_1 \cup \mathcal{D}_2$. Since $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ and $\mathcal{D}_1$ and $\mathcal{D}_2$ are subsets of $\mathcal{D}$ we know $\overline{\Gamma}{\downarrow}_{\mathcal{D}_i \cup \{\mathrm{n}\}} = \overline{\Gamma}{\downarrow}_{\mathcal{D}_i} \subseteq \overline{\Gamma}{\downarrow}_{\mathcal{D}}$ and $\overline{\Delta}{\downarrow}_{\mathcal{D}_i \cup \{\mathrm{n}\}} = \overline{\Delta}{\downarrow}_{\mathcal{D}_i} \subseteq \overline{\Delta}{\downarrow}_{\mathcal{D}}$ for $i \in \{1, 2\}$. By monotonicity we obtain

    $$\vDash \overline{\Gamma}{\downarrow}_{\mathcal{D}}, A \supset \overline{\Delta}{\downarrow}_{\mathcal{D}} \qquad \vDash \overline{\Gamma}{\downarrow}_{\mathcal{D}}, B \supset \overline{\Delta}{\downarrow}_{\mathcal{D}}.$$

Hence $\vDash \overline{\Gamma}{\downarrow_{\mathcal{D}}}, A \vee B \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ and thus $\mathcal{D}, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$, which is equivalent to $\mathcal{D}_1, \mathcal{D}_2, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$, is valid.

- $\dfrac{\mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}}{\mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}} (\cdot \vee \supset)\ \mathrm{n} \notin \mathcal{D}$

  Suppose $\mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ is valid. Since $\mathrm{n} \notin \mathcal{D}$ we know from the definition of validity $\vDash \overline{\Gamma}{\downarrow_{\mathcal{D}}} \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$. By monotonicity we also know $\vDash \overline{\Gamma}{\downarrow_{\mathcal{D}}}, A \vee B \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$. Thus, whether m is an element of $\mathcal{D}$ or not, we know that $\mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ is valid.

□

Definition 2.11 (semantically invertible rule)
A rule is called *semantically invertible* if the validity of the conclusion implies the validity of its premises.

Some of the rules of CPC2 are semantically invertible while the two branching rules are only semantically invertible if the principal formula and the active formulas in the premises carry the same label and the use-set in the premises is the same as the one in the conclusion.

Lemma 2.12 (semantically invertible rules of CPC2)

1. The CPC2-rules $(\neg\ \supset)$, $(\supset\ \neg)$, $(\wedge\ \supset)$ and $(\supset\ \vee)$ are semantically invertible.
2. If $\vDash \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{n}}{\vee} B \supset \overline{\Delta}$ then $\vDash \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $\vDash \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$.
3. If $\vDash \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, A \overset{\mathrm{n}}{\wedge} B$ then $\vDash \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}$ and $\vDash \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{B}$.

Proof.

1. Suppose $\vDash \mathcal{D}; \overline{\Gamma}, \neg \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and let $\Pi := \overline{\Gamma}{\downarrow_{\mathcal{D}}}$, $\Sigma := \overline{\Delta}{\downarrow_{\mathcal{D}}}$.

   $\mathrm{n} \notin \mathcal{D}$**:** Then obviously $\vDash \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}$.

   $\mathrm{n} \in \mathcal{D}$**:** Then we know $\vDash \Pi, \neg A \supset \Sigma$ from which we immediately obtain $\vDash \Pi \supset \Sigma, A$, i.e. $\vDash \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}$.

   The proof of the other rules is accordingly.

2. Suppose $\vDash \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{n}}{\vee} B \supset \overline{\Delta}$ and let $\Pi := \overline{\Gamma}{\downarrow_{\mathcal{D}}}$, $\Sigma := \overline{\Delta}{\downarrow_{\mathcal{D}}}$.

   $\mathrm{n} \notin \mathcal{D}$**:** As in point 1.

   $\mathrm{n} \in \mathcal{D}$**:** Then $\vDash \Pi, A \vee B \supset \Sigma$.

This implies $\models \Pi, A \supset \Sigma$ and $\models \Pi, B \supset \Sigma$, hence $\models \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\text{n}}{A}$ and $\models \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\text{n}}{B}$.

3. Analog to the proof of the previous rule.

□

Theorem 2.13 (completeness of CPC2)
If a labeled sequent $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is valid then it is deducible in CPC2.

Proof. Let $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ be a valid labeled sequent. We show our claim by induction on $\text{len}(\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta})$.

- $\text{len}(\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}) = |\overline{\Gamma}| + |\overline{\Delta}|$.

  Then $\overline{\Gamma}$ and $\overline{\Delta}$ contain only atomic formulas. Since $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is valid we know that $\overline{\Gamma}{\downarrow_{\mathcal{D}}} \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ is valid and must be an axiom. Therefore $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is an axiom, too and thus deducible in CPC2.

- $\text{len}(\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}) > |\overline{\Gamma}| + |\overline{\Delta}|$.

  Then there exists a non-atomic formula $\overset{\text{n}}{A}$ in $\overline{\Gamma} \cup \overline{\Delta}$. We distinguish on the form and position of $\overset{\text{n}}{A}$.

  – $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta} = \mathcal{D}; \overline{\Gamma}', \neg\overset{\text{n}}{B} \supset \overline{\Delta}$.

    Then by Lemma 2.12 $\mathcal{D}; \overline{\Gamma}' \supset \overline{\Delta}, \overset{\text{n}}{B}$ is valid and by induction hypothesis deducible in CPC2. Using $(\neg \supset)$ we can deduce our sequent.

$$\frac{\mathcal{D}; \overline{\Gamma}' \supset \overline{\Delta}, \overset{\text{n}}{B}}{\mathcal{D}; \overline{\Gamma}', \neg\overset{\text{n}}{B} \supset \overline{\Delta}}(\neg\supset)$$

  – $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta} = \mathcal{D}; \overline{\Gamma}', B \overset{\text{n}}{\vee} C \supset \overline{\Delta}$.

    Then by Lemma 2.12 $\mathcal{D}; \overline{\Gamma}', \overset{\text{n}}{B} \supset \overline{\Delta}$ and $\mathcal{D}; \overline{\Gamma}', \overset{\text{n}}{C} \supset \overline{\Delta}$ are valid and by induction hypothesis deducible in CPC2. If $\text{n} \notin \mathcal{D}$ then we use $(\cdot\vee \supset)$ to deduce our sequent, otherwise we use $(\vee \supset)$ to deduce it.

$$\frac{\mathcal{D}; \overline{\Gamma}', \overset{\text{n}}{B} \supset \overline{\Delta}}{\mathcal{D}; \overline{\Gamma}', B \overset{\text{n}}{\vee} C \supset \overline{\Delta}}(\cdot\vee\supset) \qquad \frac{\mathcal{D}; \overline{\Gamma}', \overset{\text{n}}{B} \supset \overline{\Delta} \quad \mathcal{D}; \overline{\Gamma}', \overset{\text{n}}{C} \supset \overline{\Delta}}{\mathcal{D}; \overline{\Gamma}', B \overset{\text{n}}{\vee} C \supset \overline{\Delta}}(\vee\supset)$$

  The other cases can be proved analogously.

□

$$\frac{\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A}}{\mathcal{D};\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\to\mathbf{B}}\supset\overline{\Delta}}(\cdot\to\supset)^a \qquad \frac{\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A}\supset\overline{\Delta},\overset{\mathrm{n}}{B}}{\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{\mathbf{A}\to\mathbf{B}}}(\supset\to)$$

$$\frac{\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{B}\supset\overline{\Delta}}{\mathcal{D};\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\to\mathbf{B}}\supset\overline{\Delta}}(\to\cdot\supset)^a$$

$$\frac{\mathcal{D}_1,\mathrm{n};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A} \quad \mathcal{D}_2,\mathrm{n};\overline{\Gamma},\overset{\mathrm{n}}{B}\supset\overline{\Delta}}{\mathcal{D}_1,\mathcal{D}_2,\mathrm{m},\mathrm{n}^*;\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\to\mathbf{B}}\supset\overline{\Delta}}(\to\supset)^b$$

$$\frac{\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B}\supset\overline{\Delta}}{\mathcal{D};\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\leftrightarrow\mathbf{B}}\supset\overline{\Delta}}(\overset{\mathrm{ll}}{\leftrightarrow}\supset)^a \qquad \frac{\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A}\supset\overline{\Delta},\overset{\mathrm{n}}{B}}{\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{A}\leftrightarrow\mathbf{B}}}(\supset\overset{\mathrm{lr}}{\leftrightarrow})^a$$

$$\frac{\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B}}{\mathcal{D};\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\leftrightarrow\mathbf{B}}\supset\overline{\Delta}}(\overset{\mathrm{rr}}{\leftrightarrow}\supset)^a \qquad \frac{\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{B}\supset\overline{\Delta},\overset{\mathrm{n}}{A}}{\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{A}\leftrightarrow\mathbf{B}}}(\supset\overset{\mathrm{rl}}{\leftrightarrow})^a$$

$$\frac{\mathcal{D}_1,\mathrm{n};\overline{\Gamma},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B}\supset\overline{\Delta} \quad \mathcal{D}_2,\mathrm{n};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B}}{\mathcal{D}_1,\mathcal{D}_2,\mathrm{m},\mathrm{n}^*;\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\leftrightarrow\mathbf{B}}\supset\overline{\Delta}}(\leftrightarrow\supset)^b$$

$$\frac{\mathcal{D}_1,\mathrm{n};\overline{\Gamma},\overset{\mathrm{n}}{A}\supset\overline{\Delta},\overset{\mathrm{n}}{B} \quad \mathcal{D}_2,\mathrm{n};\overline{\Gamma},\overset{\mathrm{n}}{B}\supset\overline{\Delta},\overset{\mathrm{n}}{A}}{\mathcal{D}_1,\mathcal{D}_2,\mathrm{m},\mathrm{n}^*;\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{A}\leftrightarrow\mathbf{B}}}(\supset\leftrightarrow)^b$$

[a] $\mathrm{n}\notin\mathcal{D}$
[b] $\mathrm{n}\notin\mathcal{D}_1\cup\mathcal{D}_2$
* if $\mathrm{n}\in\mathrm{labels}(\overline{\Gamma}\cup\overline{\Delta})$, otherwise without n

Figure 2.11: Macro-rules of CPC2

Macro-Rules

As in CPC we define macro-rules for the connective $\to$ and $\leftrightarrow$ for more compact proofs representations. The rules are given in Figure 2.11. Again the rules for introducing an equation in the antecedent contain an implicit simplification.

2.3.3 Examples

We would like to illustrate the calculus CPC2 a little bit more by giving three different proofs for the same labeled sequent.

The labeled sequent we would like to give proofs of is

$$\mathcal{D}; p \wedge \overset{0}{r} \vee q, \mathbf{r} \overset{1}{\to} \mathbf{s} \supset \overset{2}{p}, \overset{3}{q}$$

In order to have a reference when explaining the proofs, we first give a complete proof tree having no concrete use-sets

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{D}_{11}; \overset{5}{p}, \overset{5}{r} \supset \overset{2}{p}, \overset{3}{q}, \overset{4}{r}}(\mathsf{id})}{\mathcal{D}_{11}; p \wedge \overset{5}{r} \supset \overset{2}{p}, \overset{3}{q}, \overset{4}{r}}(\wedge\supset) \quad \dfrac{}{\mathcal{D}_{12}; \overset{5}{q} \supset \overset{2}{p}, \overset{3}{q}, \overset{4}{r}}(\mathsf{id})}(\vee\supset) \quad \dfrac{\dfrac{\dfrac{}{\mathcal{D}_{21}; \overset{5}{p}, \overset{5}{r}, \overset{4}{s} \supset \overset{2}{p}, \overset{3}{q}}(\mathsf{id})}{\mathcal{D}_{21}; p \wedge \overset{5}{r}, \overset{4}{s} \supset \overset{2}{p}, \overset{3}{q}}(\wedge\supset) \quad \dfrac{}{\mathcal{D}_{22}; \overset{5}{q}, \overset{4}{s} \supset \overset{2}{p}, \overset{3}{q}}(\mathsf{id})}(\vee\supset)}{\mathcal{D}_1; p \wedge \overset{0}{r} \vee q \supset \overset{2}{p}, \overset{3}{q}, \overset{4}{r} \qquad \mathcal{D}_2; p \wedge \overset{0}{r} \vee q, \overset{4}{s} \supset \overset{2}{p}, \overset{3}{q}}(\to\supset)}$$
$$\mathcal{D}; p \wedge \overset{0}{r} \vee q, r \overset{1}{\to} s \supset \overset{2}{p}, \overset{3}{q}$$

The first proof makes use of the fact that $\overset{4}{s}$ is not used as principal formula in any axiom of the right proof branch. We can thus use the rule $(\to \cdot \supset)$ to deduce $r \overset{1}{\to} s$ and omit the left proof branch.

This proof is obtained with the use-check method if the left branch has been calculated and $\overset{4}{r}$ has been chosen as principal formula in the left most axiom. When proving the right branch the algorithm will notify that $\overset{4}{s}$ was not used in the proof and can thus drop the relevance information of the left branch. As a result $r \overset{1}{\to} s$ is known as not being relevant since 1 is not in the use-set of the resulting sequent.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{2,5; \overset{5}{\mathbf{p}}, \overset{5}{r}, \overset{4}{s} \supset \overset{2}{\mathbf{p}}, \overset{3}{q}}(\mathsf{id})}{2,5; \mathbf{p} \wedge \overset{5}{\mathbf{r}}, \overset{4}{s} \supset \overset{2}{p}, \overset{3}{q}}(\wedge\supset) \quad \dfrac{}{3,5; \overset{5}{\mathbf{q}}, \overset{4}{s} \supset \overset{2}{p}, \overset{3}{\mathbf{q}}}(\mathsf{id})}(\vee\supset)}{0,2,3; \mathbf{p} \wedge \overset{0}{\mathbf{r}} \vee \mathbf{q}, \overset{4}{s} \supset \overset{2}{p}, \overset{3}{q}}(\to\cdot\supset)$$
$$0,2,3; p \wedge \overset{0}{r} \vee q, \mathbf{r} \overset{1}{\to} \mathbf{s} \supset \overset{2}{p}, \overset{3}{q} \tag{1}$$

The second proof makes use of the fact that in the left most axiom $\overset{4}{r}$ is not necessarily needed as principal formula since we have the choice to use $\overset{2}{p}$ and $\overset{5}{p}$ as principal formulas. Like that the algorithm does not need to prove the right branch and can apply $(\cdot \to\supset)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{2,5; \overset{5}{\mathbf{p}}, \overset{5}{r} \supset \overset{2}{\mathbf{p}}, \overset{3}{q}, \overset{4}{r}}(\mathsf{id})}{2,5; \mathbf{p} \wedge \overset{5}{\mathbf{r}} \supset \overset{2}{p}, \overset{3}{q}, \overset{4}{r}}(\wedge\supset) \quad \dfrac{}{3,5; \overset{5}{\mathbf{q}} \supset \overset{2}{p}, \overset{3}{\mathbf{q}}, \overset{4}{r}}(\mathsf{id})}(\vee\supset)}{0,2,3; \mathbf{p} \wedge \overset{0}{\mathbf{r}} \vee \mathbf{q} \supset \overset{2}{p}, \overset{3}{q}, \overset{4}{r}}(\cdot\to\supset)$$
$$0,2,3; p \wedge \overset{0}{r} \vee q, \mathbf{r} \overset{1}{\to} \mathbf{s} \supset \overset{2}{p}, \overset{3}{q} \tag{2}$$

The last proof does not follow the use-check method but uses a use-set in the right most axiom that contains the label 4 and is thus not minimal.

By using $\overset{4}{r}$ in the leftmost axiom as principal formula, we get thus a proof similar to the one that would be done in CPC.

$$\dfrac{\dfrac{\dfrac{}{4,5;\mathbf{p}\overset{5}{,}\mathbf{r}\overset{5}{,}\overset{4}{r}\supset\overset{2}{p},\overset{3}{q},\mathbf{r}}(\mathrm{id})}{4,5;\mathbf{p}\overset{5}{\wedge}\mathbf{r}\supset\overset{2}{p},\overset{3}{q},\overset{4}{r}}(\wedge\supset) \quad \dfrac{}{3,5;\mathbf{q}\overset{5}{\supset}\overset{2}{p},\overset{3}{\mathbf{q}},\overset{4}{r}}(\mathrm{id})}{0,3,4;\mathbf{p}\wedge\mathbf{r}\overset{0}{\vee}\mathbf{q}\supset\overset{2}{p},\overset{3}{q},\overset{4}{r}}(\vee\supset) \quad \dfrac{\dfrac{\dfrac{}{2,5;\mathbf{p}\overset{5}{,}\mathbf{r}\overset{5}{,}\overset{4}{s}\supset\mathbf{p}\overset{2}{,}\overset{3}{q}}(\mathrm{id})}{2,5;\mathbf{p}\overset{5}{\wedge}\mathbf{r},\overset{4}{s}\supset\overset{2}{p},\overset{3}{q}}(\wedge\supset) \quad \dfrac{}{3,4,5;\mathbf{q}\overset{5}{,}\overset{4}{s}\supset\overset{2}{p},\overset{3}{\mathbf{q}}}(\mathrm{id})}{0,2,3,4;\mathbf{p}\wedge\mathbf{r}\overset{0}{\vee}\mathbf{q},\overset{4}{s}\supset\overset{2}{p},\overset{3}{q}}(\vee\supset)}{0,1,2,3;p\wedge\overset{0}{r}\vee q,\mathbf{r}\overset{1}{\to}\mathbf{s}\supset\overset{2}{p},\overset{3}{q}}(\to\supset) \tag{3}$$

2.4 Refutation Search

Since CPRC is converse to CPC, a sequent $\Gamma \supset \Delta$ is refutable if and only if the sequent $\Gamma \supset \Delta$ is not valid. We can therefore use the CPC prover with its enhancements to verify whether a sequent is refutable.

By adding an additional axiom to CPC2 we obtain a calculus which can be used to prove and refute sequents.

Let CPC3 be the calculus with the rules of CPC2 and the additional axiom

$$\dfrac{}{\emptyset;\overline{\Gamma}\supset\overline{\Delta}}(\not\emptyset)$$

where $\overline{\Gamma}{\downarrow_{\mathbb{N}}} \subseteq \mathcal{V} \cup \{\top\}$, $\overline{\Delta}{\downarrow_{\mathbb{N}}} \subseteq \mathcal{V} \cup \{\bot\}$ and $\overline{\Gamma}{\downarrow_{\mathbb{N}}} \cap \overline{\Delta}{\downarrow_{\mathbb{N}}} = \emptyset$.

A sequent $\Gamma \supset \Delta$ is then defined to be refutable if $\emptyset;\overset{\mathrm{m}}{\Gamma} \supset \overset{\mathrm{m}}{\Delta}$ is deducible in CPC3 for any label m.

2.5 Experimental Results

In this section we have a look at two scalable formulas. One of them, the pigeonhole formula, is used to investigate the efficiency of use-check. The other formula, the one of Urquhart, is used for two aspects. On the one hand we use it to measure the overhead produced by the use-check mechanism. On the other hand we use it to show that for proof search it is advantageous to operating over an extended language $\mathcal{L}^+$ in which the connectives $\to$ and $\leftrightarrow$ are defined to be part of the language.

We used Debian Linux 4.0 on an AMD Sempron 2600+ and 1.5 GB RAM to run our tests.

	without use-check		with use-check	
n	axioms	time (h:m:s)	axioms	time (m:s)
1	2	< 0:01	2	< 0:01
2	48	< 0:01	14	< 0:01
3	33.531	< 0:01	136	< 0:01
4	3.630.464.872	8:17:02	1.601	< 0:01
5			21.479	< 0:01
6			298.579	0:05
7			4.523.139	1:30
8			71.163.129	29:07

Table 2.1: Figures of proving pigeonhole(n) with and without use-check

2.5.1 The Pigeonhole Principle

The pigeonhole principle states that if there are m objects that are to be put into n boxes and n is less than m, then there is a box containing at least two objects. The name of the principle is inspired by pigeons that sit in their holes.

The problem we investigate is about $n+1$ pigeons and n holes. Formally we use a language $\mathcal{L}^+(\mathcal{V})$ with $\mathcal{V} = \{p_{ij} : 1 \leq i \leq n+1 \text{ and } 1 \leq j \leq n\}$. The interpretation of the propositional variable p_{ij} thereby states whether pigeon i sits in hole j or not. We use the following formula to specify the principle.

$$\text{pigeonhole}(n) := \bigwedge_{i=1}^{n+1} \bigvee_{j=1}^{n} p_{ij} \rightarrow \bigvee_{j=1}^{n} \bigvee \{p_{ij} \wedge p_{i'j} : 1 \leq i < i' \leq n+1\}$$

In this formula the left part of the implication states for each pigeon i that it sits in one of the holes while the right part of the implication states that two different pigeons sit in the same hole. This specification of the pigeonhole principle seems a bit inexact since the left part of the implication evaluates also to true if some pigeons sit in more than one hole. But because we also cover the case where each pigeon sits in exactly one hole, the principal is specified correctly.

In table 2.1 the time used to prove pigeonhole(n) together with the number of axioms used during proof search are given. The number of axioms thereby is equivalent to the number of branchings + 1. We see that with use-check pigeonhole(4) can be proved in less than a second while without use-check it takes more than eight hours. Apparently this is because with use-check we are able to reduce the number of branchings dramatically. To prove pigeonhole(4) for example we reach less that 2.000 axioms in the prover

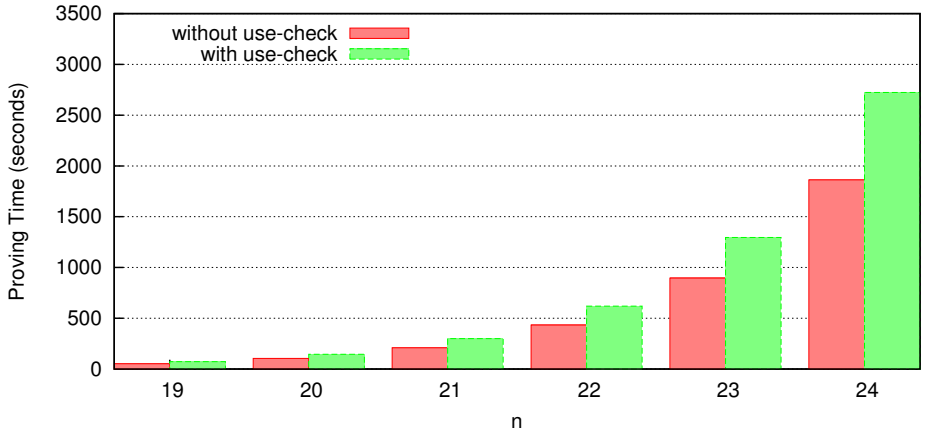


Figure 2.12: Proving time of urquhart(n) with and without use-check

that implements use-check while in the prover without use-check we end up with over 3 billion axioms.

2.5.2 Urquhart's Formula

The next formula we want to investigate is the one proposed by Urquhart [26]. It consists of nested equivalences and is of the following form.

$$\text{urquhart}(n) := p_1 \leftrightarrow p_2 \leftrightarrow \ldots \leftrightarrow p_n \leftrightarrow p_1 \leftrightarrow p_2 \leftrightarrow \ldots \leftrightarrow p_n$$

Since $((A \leftrightarrow B) \leftrightarrow C)$ is equivalent to $(A \leftrightarrow (B \leftrightarrow C))$ we know that urquhart(n) is equivalent to $(p_1 \leftrightarrow \ldots \leftrightarrow p_n) \leftrightarrow (p_1 \leftrightarrow \ldots \leftrightarrow p_n)$ and thus obviously a tautology.

The formula is well suited to test the overhead produced by use-check for two reasons. Firstly because the prover is unable to omit any branch when proving this formula and secondly because the prover has to deal only with branching rules.
The time used to prove urquhart(n) is shown in figure 2.12. The overhead produced by use-check for urquhart(n) is about 50%. This seems rather much but compared to the speedup we can gain with use-check it is a small price to pay.

The formula is also well suited to show that operating over the extended language $\mathcal{L}^+$ is reasonable. Suppose urquhart2(n) is the formula urquhart(n) in which we replace all formulas of the form $A \leftrightarrow B$ with $(A \rightarrow B) \wedge (B \rightarrow A)$. If we compare the length of the two formulas we have $\text{len}(\text{urquhart}(n)) = 4n-1$

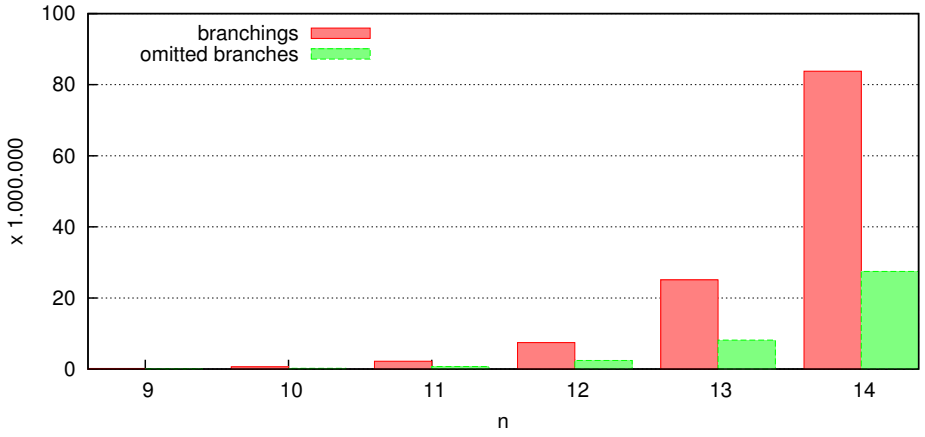


Figure 2.13: Proving statistics of urquhart2(n)

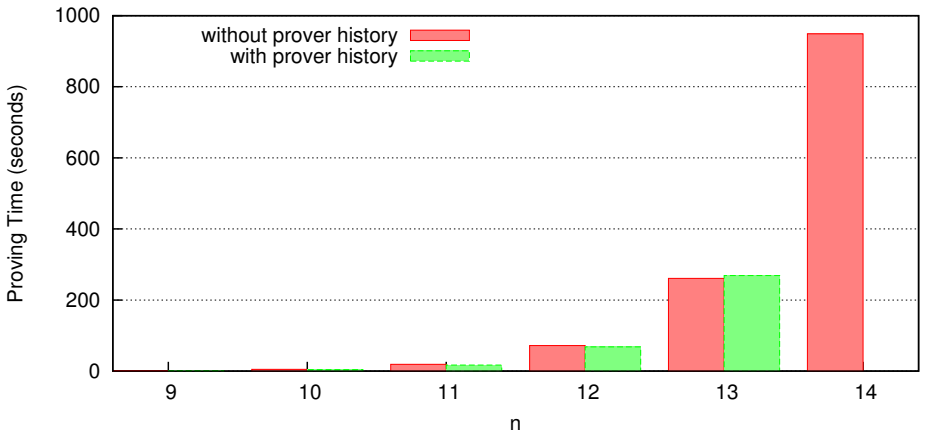


Figure 2.14: Proving time of urquhart2(n)

while len(urquhart2(n)) $= 3 \cdot 4^n - 5$. Hence by replacing the connective $\leftrightarrow$ with an equivalent definition the length of the formula grows exponentially.

In contrast to when proving urquhart(n), use-check helps us to omit branches when proving urquhart2(n). For urquhart2(14) for example we can omit proving the second branch in about 30% of the roughly 80 million processed branching rules (see Figure 2.13). However, this is not sufficient to prove urquhart2(n) in about the same time as urquhart(n), since to prove urquhart(14) we only need less than 250.000 branchings. In Figure 2.14 you see the time used to prove urquhart2(n). While urquhart(22) can be proved in ten minutes, we already need 15 minutes to prove urquhart2(14). Moreover, because of the exponentially growing formula length, we already run out of memory when trying to prove urquhart2(15).

It seems odd that the number of branchings in the proofs of the two equivalent formulas differ so much. The difference bases mainly on the chosen proving strategy, the implicit simplification that we have put into the rule $(\leftrightarrow\supset)$ (s. page 14) and the fact that the principal formulas in the identity axioms must be atomic.

To obtain proofs for $\Gamma, A \leftrightarrow B \supset \Delta$ and $\Gamma, (A \to B) \wedge (B \to A) \supset \Delta$ with similar complexity we would have to allow identity axioms with non-atomic principal formulas. Furthermore our prover would then need to find a proof of the following form.

$$\dfrac{\dfrac{\dfrac{\Gamma \supset \Delta, A, B \quad \dfrac{}{\Gamma, A \supset \Delta, A}(\mathsf{id})}{\Gamma, \mathbf{B} \to \mathbf{A} \supset \Delta, A}(\to\supset) \quad \dfrac{\dfrac{}{\Gamma, B \supset \Delta, B}(\mathsf{id}) \quad \Gamma, B, A \supset \Delta}{\Gamma, B, \mathbf{B} \to \mathbf{A} \supset \Delta}(\to\supset)}{\Gamma, \mathbf{A} \to \mathbf{B}, B \to A \supset \Delta}(\to\supset)}{\Gamma, (\mathbf{A} \to \mathbf{B}) \wedge (\mathbf{B} \to \mathbf{A}) \supset \Delta}(\wedge\supset)$$

However, our prover is working in a way that after applying a rule backwards we continue by processing the active formulas in the premises. Therefore we process A and B before $B \to A$ and obtain thus a proof of the following form.

$$\dfrac{\dfrac{\Gamma, B \to A \supset \Delta, \mathbf{A} \quad \Gamma, \mathbf{B}, B \to A \supset \Delta}{\Gamma, \mathbf{A} \to \mathbf{B}, B \to A \supset \Delta}(\to\supset)}{\Gamma, (\mathbf{A} \to \mathbf{B}) \wedge (\mathbf{B} \to \mathbf{A}) \supset \Delta}(\wedge\supset)$$

Unless A and B are atomic we will thus not encounter the sequents $\Gamma, A \supset \Delta, A$ or $\Gamma, B \supset \Delta, B$ during proof search.

Using a Prover History

In the example above we can not prevent that the formula A is processed first if we want to hold on to our proving strategy. But we have the option to remember that A was in the succedent during proof search. If we later process $B \to A$ then the second premise of that backward application will have A in its antecedent. Before proving that premise we then first verify whether A ever appeared in the succedent of the proof, which will be the case in our example. Therefore we can immediately return with success for that premise.

In general this means that we have to remember all formulas that appear in the sequents of our proof during proof search. To establish this we add to our prover a sequent holding the so called prover history. Whenever a rule is applied backwards, we add the active formulas in the premise to the prover

history.[2] Before proceeding with processing the side formulas we then first check whether the same formula is already present on the other side of the prover sequent. If it is, we know that the premise is provable and can stop proof search with success. The formula we check for in the prover history is thereby generally not present on the other side of the sequent that is still to prove. We can therefore say that this mechanism goes further than allowing identity axioms with non-atomic principal formulas. To illustrate this approach we have a closer look at the example from above.

We initially set the prover history to the sequent we want to prove, i.e. $\Gamma, (A \to B) \wedge (B \to A) \supset \Delta$. When applying $(\wedge \supset)$ backwards we first check whether $A \to B$ or $B \to A$ is in the succedent of the prover history. If this is the case then we have found an axiom and may return with success. If this is not the case then we add $A \to B$ and $B \to A$ to the antecedent of the prover history and proceed with processing the formula $A \to B$.
For the left branch we first check whether A is present in the antecedent of the prover history and return with success if it is. If it is not we add A to the succedent of the prover history and proceed by processing A. Later we will eventually process the formula $B \to A$ in that branch. In the second premise of that backward rule application we will then be successful in testing whether A is in the succedent of the prover history. Hence processing A can then be omitted and we can return with success.

We have implemented such a prover history using hashed sets of formulas. Although we could heavily reduce the number of branchings used in proof search (see Figure 2.15), the results were discouraging. Because of the additional memory needed by the prover history, we already ran out of memory when trying to prove urquhart2(14). Furthermore the overhead produced by the prover history negated the advantage of having less branchings and we ended up with about the same proving times as with the prover without prover history (see Figure 2.14).

This example makes it clear that operating over an extended language $\mathcal{L}^+$ can be crucial for the time the prover needs for certain problems. As a consequence we may state that a compact notation of formulas can result in more efficient proof search.

[2] For branching rules we obtain different prover histories for the right and left premise

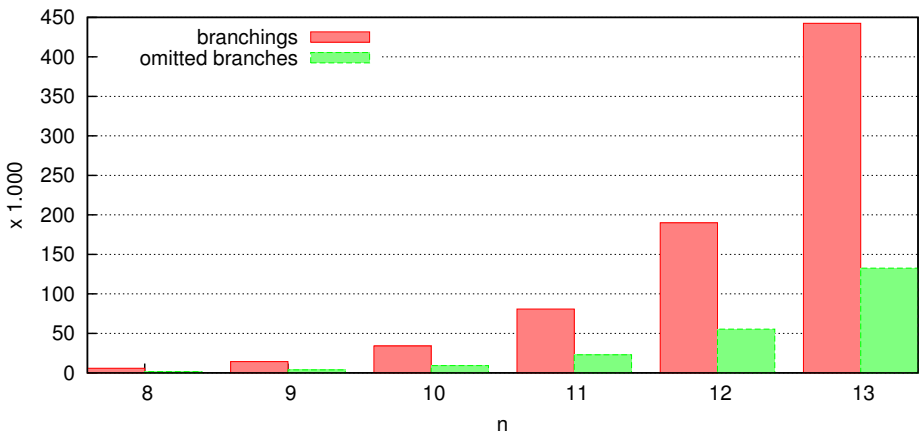


Figure 2.15: Proving statistics of urquhart2(n)

Chapter 3

Propositional Circumscription Logic

We start this chapter with an example showing the main idea of circumscription and continue with the definition of propositional circumscription logic (PCL). Then we inspect several kind of changes under which the so called minimal models remain minimal. We continue by showing how a circumscription theory can be expressed in CPL such that proving can be done using classical propositional logic. Afterward we recall the sequent calculus for PCL by Bonatti and Olivetti [2] and close with the proof of the introductory example.

3.1 Introductory Example

We are going to introduce circumscription on one of the most famous examples in non-monotonic logic: The case of "Tweedy" the penguin. The example is about the fact that birds generally fly. Nobody would deduce that a bird can not fly unless it is known that the bird is in some way abnormal. For example if the bird is a penguin.

We model this situation using a variable abn indicating that the object in question is abnormal. Our knowledge base Γ consists then of three formulas, the first states that a bird that is not abnormal can fly, the second states that a penguin is a bird and can not fly, the third states that the object in question is a bird.

$$\Gamma := \{\mathsf{bird} \wedge \neg\mathsf{abn} \rightarrow \mathsf{fly}, \mathsf{penguin} \rightarrow \neg\mathsf{fly} \wedge \mathsf{bird}, \mathsf{bird}\}$$

Using classical logic the only fact we can deduce is that a penguin is abnormal, it is not possible to apply general knowledge that our bird can fly.

The idea of circumscription logic is to minimize abnormal objects by considering only those models that are minimal regarding abnormal properties. In our simple example this is rather easy. Let us look at the four different models of Γ.

	M_0	M_1	M_2	M_3
bird	1	1	1	1
abn	0	1	1	1
penguin	0	0	0	1
fly	1	0	1	0

Now model M_0 is the only model where abn is not valid. Regarding the variable abn it is thus the only minimal model and hence the only model that is considered for deduction in circumscription logic. Since fly holds in M_0 we thus conclude in circumscription logic that the object in question can fly.

In circumscription logic we have the further possibility to restrict the comparison of two models. We do this by requiring that two models are only comparable if they do not differ in their interpretation of certain variables. Suppose we further require in our example that the models are only comparable if they do not differ in their interpretation of the variable penguin. Then M_3 is no longer comparable with the other models. We thus end up with the two minimal models M_0 and M_3. As a consequence we can no longer deduce fly. This approach reflects the situation where we model two kind of worlds, one without penguins and one containing only penguins.

Our example also illustrates that circumscription is non-monotonic. Consider the case where all models are comparable. If we add penguin to Γ — that is we add the information that our object is a penguin — then the only model of Γ is M_3. Thus, although we are given more information than before, we can no longer deduce fly.

3.2 Propositional Circumscription Logic

Propositional circumscription logic is defined semantically on the notion of minimal models. This notion is based on a partial order on the set of interpretations of $\mathcal{L}$. Among the several known forms of circumscription we present in this section the most common form where variables are allowed to vary and more than one propositional variable is minimized.

The partial order on the interpretations of $\mathcal{L}$ is based on a set P of propositional variables that are to minimize and a set R of propositional variables in which the interpretations may not vary in order to be comparable. The two sets P and R implicitly define the set Q of propositional variables which are allowed to be interpreted differently by two comparable interpretations.

Definition 3.1 ($I \leq_{P;R} J$)
For $P \subset \mathcal{V}$ and $R \subset \mathcal{V}$ with $P \cap R = \emptyset$ we define a partial order $\leq_{P;R}$ on the set of interpretations of $\mathcal{L}(\mathcal{V})$.

$$I \leq_{P;R} J \quad :\Longleftrightarrow \quad P \cap I \subseteq P \cap J \text{ and } R \cap I = R \cap J$$

We write $I <_{P;R} J$ if $I \leq_{P;R} J$ and not $J \leq_{P;R} I$.

Elements of P are called *minimized* variables, those of R are called *fixed* variables. All other propositional variables, that is elements of $Q := \mathcal{V}_{\mathcal{L}} \setminus (P \cup R)$, are called *varied* variables.

The notion of a $(P;R)$-minimal model of a set of formulas Γ is based on the partial order $\leq_{P;R}$. The central definition is that of a $(P;R)$-minimally entailed formula. It is defined analogous to the notion of a logical consequence in CPL.

Definition 3.2 ($(P;R)$-minimal model)
A model M of a set of formulas Γ is called a $(P;R)$-*minimal model* of Γ if there is no model N of Γ such that $N <_{P;R} M$. We then write $M \models_{P;R} \Gamma$. Accordingly we write $M \not\models_{P;R} \Gamma$ to denote that M is not a $(P;R)$-minimal model of Γ.

Definition 3.3 (minimal entailment)
A set of formulas Γ $(P;R)$-*minimally entails* a formula A if A is valid in all $(P;R)$-minimal models of Γ. We then write $\Gamma \mid\!\!\sim_{P;R} A$.

If R is empty we normally write $\leq_P$, $<_P$, $\models_P$ and $\mid\!\!\sim_P$, respectively and speak of P-minimal models.

3.2.1 Some Properties of Minimal Models

The property of being $(P;R)$-minimal for a multiset of formulas Γ does not change for a model M if a formula that is valid in M is added to Γ.

Lemma 3.4 (add valid formulas to the knowledge base)
If $M \models_{P;R} \Gamma$ and $M \models A$ then $M \models_{P;R} A, \Gamma$.

Proof. Let M be a $(P;R)$-minimal model of Γ with $M \models A$, then $M \models A, \Gamma$. Suppose there exists a model N of A, Γ such that $N <_{P;R} M$. Then also $N \models \Gamma$ which contradicts to $M \models_{P;R} \Gamma$. □

If Γ entails a propositional variable p or its negation $\neg p$ then p is of no importance for the property of being $(P;R)$-minimal. This is because p evaluates to the same value in every model of Γ and has thus no influence on the operator $\leq_{P;R}$.

Corollary 3.5 (extend minimal or fixed variables)
Let P and R be two sets of variables such that $P \cap Q = \emptyset$ and let $p \notin P \cup R$. Furthermore let Γ be a (multi)set of formulas such that $\Gamma \Vdash p$ or $\Gamma \Vdash \neg p$. Then the following holds:

$$M \models_{P;R} \Gamma \qquad \Longleftrightarrow \qquad M \models_{P,p;R} \Gamma \qquad \Longleftrightarrow \qquad M \models_{P;R,p} \Gamma.$$

And as an immediate consequence we have for every formula A under the given preconditions

$$\Gamma \Vdash_{P;R} A \qquad \Longleftrightarrow \qquad \Gamma \Vdash_{P,p;R} A \qquad \Longleftrightarrow \qquad \Gamma \Vdash_{P;R,p} A.$$

The following two lemmas are special cases of the corollary above but use weaker preconditions. Lemma 3.6 states that a $(P, p; R)$-minimal model M of Γ is also $(P; R)$-minimal if p is valid in M.
Lemma 3.7 says that the set of $(P; R)$-minimal models of Γ is contained in the set of $(P; R, r)$-minimal models of Γ. This is because by having less fixed variables we increase the number of comparable models. Thus the number of minimal models can decrease.

Lemma 3.6 (drop minimized variables)
If $M \models_{P,p;R} \Gamma$ and $M \models p$ then $M \models_{P;R} \Gamma$.

Proof. Let $M \models_{P;R} \Gamma$ (1) and $M \models p$ (2)
Suppose M is not $(P; R)$-minimal for Γ. Then there exists a model N of Γ such that $N <_{P;R} M$, that is $N \cap P \subsetneq M \cap P$. We know

$$N \cap (P \cup \{p\}) \subseteq (N \cap P) \cup \{p\} \subsetneq (M \cap P) \cup \{p\} \overset{(2)}{=} M \cap (P \cup \{p\})$$

Thus $N <_{P,p;R} M$ which contradicts to (1). □

Lemma 3.7 (add fixed variables)
If $M \models_{P;R} \Gamma$ and $r \notin P$ then $M \models_{P;R,r} \Gamma$.

And as an immediate consequence we have: If $\Gamma \Vdash_{P;R,r} A$ then $\Gamma \Vdash_{P;R} A$.

Proof. Suppose $M \models_{P;R} \Gamma$ $(*)$ and $r \notin P$. We distinguish to cases.

1. $r \in R$: Then trivially $M \models_{P;R,r} \Gamma$.
2. $r \notin R$: Suppose $M \not\models_{P;R,r} \Gamma$.
   Then there exists a model N of Γ with $N <_{P;R,r} M$, i.e. $N \cap P \subsetneq M \cap P$ and $N \cap (R \cup \{r\}) = M \cap (R \cup \{r\})$. Thus $N \cap R = M \cap R$ and hence $N <_{P;R} M$ which contradicts to $(*)$.

□

The following corollary is a consequence of Lemma 3.4 and Corollary 3.5 and shows how to drag out a minimized variable into Γ.

Corollary 3.8 (drag out minimized variables)

1. If $M \vDash_{P,p;R} \Gamma$ and $M \vDash p$ then $M \vDash_{P;R} p, \Gamma$.
2. If $M \vDash_{P,p;R} \Gamma$ and $M \vDash \neg p$ then $M \vDash_{P;R} \neg p, \Gamma$.

Proof.

1. Suppose $M \vDash_{P,p;R} \Gamma$ and $M \vDash p$. Then we obtain $M \vDash_{P,p;R} \Gamma, p$ with Lemma 3.4 and with Corollary 3.5 we get $M \vDash_{P;R} \Gamma, p$.
2. Analogous to 1.

□

It is also possible to drag propositional variables from Γ into the set of minimized variables, but only if they are negative.

Lemma 3.9 (drag in minimized variables)
If $M \vDash_{P;R} \neg p, \Gamma$ and $p \notin R$ then $M \vDash_{P,p;R} \Gamma$.

Proof. Let $P' := P \cup \{p\}$, $M \vDash_{P;R} \neg p, \Gamma$ $(*)$. Then $M \vDash \Gamma$ and $M \vDash \neg p$ (1). Suppose M is not a $(P'; R)$-minimal model of Γ. Then there exists a model $N \vDash \Gamma$ (2) such that $N <_{P';R} M$ (3). With (1) and (3) we obtain $N \vDash \neg p$. Using (2) we know $N \vDash \neg p, \Gamma$ which contradicts to $(*)$. □

There are two similar lemmas for dragging out fixed variables into Γ and dragging in variables from Γ into R.

Lemma 3.10 (drag out fixed variables)

1. If $M \vDash_{P;R,r} \Gamma$ and $M \vDash r$ then $M \vDash_{P;R} r, \Gamma$.
2. If $M \vDash_{P;R,r} \Gamma$ and $M \vDash \neg r$ then $M \vDash_{P;R} \neg r, \Gamma$.

Proof.

1. If $M \vDash_{P;R,r} \Gamma$ and $M \vDash r$ then we obtain $M \vDash_{P;R,r} r, \Gamma$ with Lemma 3.4. With Lemma 3.6 we then obtain $M \vDash_{P;R} r, \Gamma$.
2. Analogous to 1.

□

Lemma 3.11 (drag in fixed variables)

1. If $M \vDash_{P;R} \Gamma, r$ and $r \notin P$ then $M \vDash_{P;R,r} \Gamma$.
2. If $M \vDash_{P;R} \Gamma, \neg r$ and $r \notin P$ then $M \vDash_{P;R,r} \Gamma$.

Proof.

1. Let $M \vDash_{P;R} \Gamma, r$ (1), $r \notin P$ and suppose that $M \nvDash_{P;R,r} \Gamma$. Then there exists a model N of Γ such that $N <_{P;R,r} M$. We know $N \cap (R \cup \{r\}) = M \cap (R \cup \{r\})$, thus $N \vDash \Gamma, r$ which contradicts to (1).
2. Analogous to 1.

□

3.2.2 Syntactic Definition

Propositional circumscription can also be defined syntactically. To obtain the minimal models of a formula A, we define a set of formulas that contains A and formulas to filter out the non-minimal models of A. The models of that set are then exactly the minimal models of A. We first illustrate this mechanism on three examples.

Example With One Minimized Variable

Let A be a formula, p be the variable that is to minimize and let the set of varied variables be empty, i.e. $R = \text{vars}(A) \setminus \{p\}$. A model M of A is $(p; R)$-minimal in two cases: If it does not contain p or if it contains p and if the interpretation $M' := M \setminus \{p\}$ is not a model of A. It is easy to see that the following formula expresses this requirement.

$$B := p \rightarrow \neg A[\bot/p]$$

Hence the models of $\{A, B\}$ are the $(p; R)$-minimal models of A.

Example With Two Minimized Variables

The next example is similar to the previous but with two minimized variables $P := \{p_0, p_1\}$ and no varied variables, i.e. $R := \text{vars}(A) \setminus P$. A model of A may contain p_0 and p_1, either p_0 or p_1, or neither p_0 nor p_1. According to these cases we define the following formulas.

$$B_{11} := p_0 \wedge p_1 \rightarrow \neg A[\top/p_0, \bot/p_1] \wedge \neg A[\bot/p_0, \top/p_1] \wedge \neg A[\bot/p_0, \bot/p_1]$$
$$B_{10} := p_0 \wedge \neg p_1 \rightarrow \neg A[\bot/p_0, \bot/p_1]$$
$$B_{01} := \neg p_0 \wedge p_1 \rightarrow \neg A[\bot/p_0, \bot/p_1]$$
$$B_{00} := \neg p_0 \wedge \neg p_1 \rightarrow \top$$

Consider the formula B_{11} together with an interpretation M. If M does not contain p_0 and p_1 then it is a model of B_{11}.
If M contains p_0 and p_1 then it is only a model of B_{11} if it fulfills the right

part of the implication. It is easy to see that this is only the case if A is not valid in any interpretation M' with $M' <_{P;R} M$.
B_{11} thus filters out those models of $p_0 \wedge p_1$ for which a lesser model exists in which A is valid.

The formulas B_{10} and B_{01} do a similar job for the models of $p_0 \wedge \neg p_1$ and $\neg p_0 \wedge p_1$, respectively. The formula B_{00} is only given for the sake of completeness, since a model that fulfills $\neg p_0 \wedge \neg p_1$ is trivially P-minimal.

Now A need not be valid in a model of B_{11}, B_{10} or B_{01}. To obtain the P-minimal models of A we must intersect the models of A with those of B_{11}, B_{10} and B_{01}. Hence the models of $\{A, B_{11}, B_{10}, B_{01}\}$ are exactly the $(P; R)$-minimal models of A.

Examples With Varied Variables

If Q is not empty, we have to encode additional information into our formulas. Since interpretations are now also comparable if they differ in their interpretation of the varied variables, we must consider every possible interpretation of the varied variables in the right side of our implications.

For the case where $P = \{p\}$, $Q = \{q\}$ and $R := \text{vars}(A) \setminus (P \cup Q)$ the formula B then looks as follows.

$$B := p \rightarrow \neg A[\bot/p, \top/q] \wedge \neg A[\bot/p, \bot/q]$$

For the case where $P = \{p_1, p_2\}$, $Q := \{q\}$ and $R := \text{vars}(A) \setminus (P \cup Q)$ our formulas then look as follows.

$$\begin{aligned}
B_{11} := \; & p_0 \wedge p_1 \rightarrow \neg A[\top/p_0, \bot/p_1, \top/q] \wedge \neg A[\top/p_0, \bot/p_1, \bot/q] \wedge \\
& \neg A[\bot/p_0, \top/p_1, \top/q] \wedge \neg A[\bot/p_0, \top/p_1, \bot/q] \wedge \\
& \neg A[\bot/p_0, \bot/p_1, \top/q] \wedge \neg A[\bot/p_0, \bot/p_1, \bot/q] \\
B_{10} := \; & p_0 \wedge \neg p_1 \rightarrow \neg A[\bot/p_0, \bot/p_1, \top/q] \wedge \neg A[\bot/p_0, \bot/p_1, \bot/q] \\
B_{01} := \; & \neg p_0 \wedge p_1 \rightarrow \neg A[\bot/p_0, \bot/p_1, \top/q] \wedge \neg A[\bot/p_0, \bot/p_1, \bot/q] \\
B_{00} := \; & \neg p_0 \wedge \neg p_1 \rightarrow \top
\end{aligned}$$

General Definition

We now give the general case. To formalize it we need some auxiliary definitions.

We write $\mathbb{B}$ to denote the set $\{0, 1\}$ and $\mathbb{B}^n$ for the set of n-tuples of $\mathbb{B}$. Given an element $\underline{i} \in \mathbb{B}^n$ we write i_k to denote the k-th element i_k of $\underline{i}$, that

is $\underline{i} := \langle i_1, i_2, \ldots, i_n \rangle$. The $<$ operator on $\mathbb{B}^n \times \mathbb{B}^n$ is defined pointwise, that is $\underline{i} < \underline{j}$ if $i_k \leq j_k$ for every $1 \leq k \leq n$ and $i_k \neq j_k$ for at least one k. The interpretation $\widehat{i}$ of $i \in \mathbb{B}$ is done straightforward, that is $\widehat{0} := \bot$ and $\widehat{1} := \top$. Given a propositional variable p we define p^0 to be the formula $\neg p$ and p^1 to be the formula p.

Definition 3.12 (CIRC(A, P, Q))
Let A be a formula and let the variables vars(A) of A be partitioned into 3 disjoint subsets P, R and Q with $P := \{p_1, \ldots, p_n\}$ and $Q := \{q_1, \ldots, q_m\}$. Then we define for every $\underline{i} \in \mathbb{B}^n$ the formula $B_{\underline{i}}(A, P, Q)$ as follows:

If Q is empty then

$$B_{\underline{i}}(A, P, Q) := \bigwedge_{k=1}^{n} p_k^{i_k} \rightarrow \bigwedge_{\underline{j} < \underline{i}} \neg A[\widehat{j_1}/p_1, \ldots, \widehat{j_n}/p_n]$$

else

$$B_{\underline{i}}(A, P, Q) := \bigwedge_{k=1}^{n} p_k^{i_k} \rightarrow \bigwedge_{\underline{j} < \underline{i}} \bigwedge_{\underline{k} \in \mathbb{B}^m} \neg A[\widehat{j_1}/p_1, \ldots, \widehat{j_n}/p_n, \widehat{k_1}/q_1, \ldots, \widehat{k_m}/q_m]$$

The circumscription theory CIRC(A, P, Q) is then defined as

$$\text{CIRC}(A, P, Q) := A \cup \left\{ B_{\underline{i}}(A, P, Q) : \underline{i} \in \mathbb{B}^n \right\}.$$

For a finite set of formulas Γ the circumscription theory is defined as the circumscription theory of its conjunction.

$$\text{CIRC}(\Gamma, P, Q) := \text{CIRC}(\bigwedge \Gamma, P, Q)$$

To understand the idea of CIRC(A, P, Q) consider the formula $B_{\underline{i}}(A, P, Q)$.

The left side of its implication is valid in all interpretations I that correspond to $\underline{i}$ regarding their interpretation of P, i.e. interpretations in which p_k is valid if $i_k = 1$ and in which $\neg p_k$ is valid if $i_k = 0$.

The right side of the implication consists of a conjunction of modified formulas basing on $\neg A$. The modification is done by substituting the variables $P \cup Q$ with $\top$ or $\bot$. Thereby we have for each $\langle \underline{j}, \underline{k} \rangle \in \mathbb{B}^n \times \mathbb{B}^m$ with $\underline{j} < \underline{i}$ a modified version of $\neg A$ in which each variable of P and Q is substituted with the value that corresponds to $\underline{j}$ and $\underline{k}$, respectively.

Now take an interpretation I that corresponds to $\underline{i}$ regarding its interpretation of P and choose $\langle \underline{j}, \underline{k} \rangle \in \mathbb{B}^n \times \mathbb{B}^m$ with $\underline{j} < \underline{i}$. Let J be the interpretation that corresponds to $\underline{j}$ and $\underline{k}$ regarding its interpretation of P and Q and that interprets the fixed variables R as I does.

Then $\neg A[\widehat{j_1}/p_1, \ldots, \widehat{j_n}/p_n, \widehat{k_1}/q_1, \ldots, \widehat{k_m}/q_m]$ is valid in I iff it is valid in J. It is easy to see that $J <_{P;R} I$. The conjunction on the right side of the implication of $B_{\underline{i}}$ contains all combinations of $\underline{j} \in \mathbb{B}^m$ with $\underline{j} < \underline{i}$ and $\underline{k} \in \mathbb{B}^m$. The formula $B_{\underline{i}}$ is therefore valid in an interpretation I if

1. I does not correspond to $\underline{i}$ regarding its interpretation of P, or
2. I corresponds to $\underline{i}$ regarding its interpretation of P and A is not valid in any interpretation $J <_{P;R} I$.

The following theorem is according to McCarthy [20] who defines the syntactical equivalence for first order logic.

Theorem 3.13 (syntactical equivalence)
Let Γ be a set of formulas and P, R, Q be a disjoint partition of vars(Γ). Then

$$\Gamma \Vdash_{P;R} A \qquad \Longleftrightarrow \qquad \mathrm{CIRC}(\Gamma, P, Q) \Vdash A$$

Proof. Let $A_\Gamma := \bigwedge \Gamma$ and the set of minimized, fixed and varied variables be $P = \{p_1, \ldots, p_n\}$, R and $Q = \{q_1, \ldots q_m\}$.
It is sufficient to show $M \vDash_{(P;R)} \Gamma$ iff $M \vDash \mathrm{CIRC}(\Gamma, P, Q)$. To do so we distinguish two cases.

$M \nvDash \Gamma$: Then we know $M \nvDash \mathrm{CIRC}(\Gamma, P, Q)$ because $\bigwedge \Gamma \in \mathrm{CIRC}(\Gamma, P, Q)$.

$M \vDash \Gamma$: Let $\underline{i} \in \mathbb{B}^n$ be such that it corresponds to the interpretation of P in M, i.e. $M \vDash \bigwedge_{k=1}^{n} p_k^{i_k}$. We distinguish two cases.
Suppose that M is not $(P;R)$-minimal for Γ. Then there exists a model N of A_Γ with $N <_{P;R} M$. Let $\underline{j} \in \mathbb{B}^n$ and $\underline{k} \in \mathbb{B}^m$ be such that they correspond to the interpretation of P and Q in N, respectively. Hence $M \vDash A_\Gamma[\widehat{j_1}/p_1, \ldots, \widehat{j_n}/p_n, \widehat{k_1}/q_1, \ldots, \widehat{k_m}/q_m]$, therefore $M \nvDash B_{\underline{i}}$ and we thus know that M is not a model of $\mathrm{CIRC}(\Gamma, P, Q)$.
Suppose that M is $(P;R)$-minimal for Γ (1). If $\underline{i}$ does not correspond to the interpretation of P in M then we know that $M \vDash B_{\underline{i}}$. So let $\underline{i}$ correspond to the interpretation of P in M and $\langle \underline{j}, \underline{k} \rangle \in \mathbb{B}^n \times \mathbb{B}^m$ with $\underline{j} < \underline{i}$. The interpretation of $A_\Gamma[\widehat{j_1}/p_1, \ldots, \widehat{j_n}/p_n, \widehat{k_1}/q_1, \ldots, \widehat{k_m}/q_m]$ in M is then equal to the interpretation of it in some $N <_{P;R} M$. Because of (1) we then know that $N \nvDash A_\Gamma[\widehat{j_1}/p_1, \ldots, \widehat{j_n}/p_n, \widehat{k_1}/q_1, \ldots, \widehat{k_m}/q_m]$ for all $\langle \underline{i}, \underline{k} \rangle \in \mathbb{B}^n \times \mathbb{B}^m$ with $\underline{j} < \underline{i}$. Hence $M \vDash \neg A_\Gamma[\widehat{j_1}/p_1, \ldots, \widehat{j_n}/p_n, \widehat{k_1}/q_1, \ldots, \widehat{k_m}/q_m]$ for all $\langle \underline{i}, \underline{k} \rangle \in \mathbb{B}^n \times \mathbb{B}^m$ with $\underline{j} < \underline{i}$ and thus $M \vDash B_{\underline{i}}$. □

3.3 A Sequent Calculus for Circumscription

In this section we introduce a sequent calculus for propositional circumscription logic of Bonatti and Olivetti [2]. To obtain a good understanding of it

$$\frac{\mathsf{CPRC} \vdash \Gamma, \neg P \supset p}{\Sigma, \mathbf{p}; \Gamma \supset_{P;\emptyset} \Delta}(\text{C1}) \qquad \frac{\Sigma, p; \Gamma \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg p \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P,\mathbf{p};R} \Delta}(\text{C3})$$

$$\frac{\mathsf{CPC} \vdash \Sigma, \Gamma \supset \Delta}{\Sigma; \Gamma \supset_{P;R} \Delta}(\text{C2}) \qquad \frac{\Sigma; \Gamma, r \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg r \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P;R,\mathbf{r}} \Delta}(\text{C4})$$

Figure 3.1: Circumscription rules of PCC

we give the original proof for soundness and a modified proof of completeness.

The calculus relies on the calculi CPC and CPRC and therefore uses CPC and CPRC sequents. Apart from that special circumscription sequents are used.

Definition 3.14 (PCC-sequent)
A PCC-sequent or circumscription sequent is a quintuple $\langle \Gamma, \Delta, \Sigma, P, R \rangle$ denoted by $\Sigma; \Gamma \supset_{P;R} \Delta$ where Γ and Δ are multisets of formulas and Σ, P and R are disjoint subsets of $\mathcal{V}$.

As usual Γ and Δ are called the *antecedent* and *succedent* and P and R are called the minimized and fixed variables. The elements of Σ are called the *constraints* of the sequent.

Definition 3.15 (valid circumscription sequent)
A PCC-sequent $\Sigma; \Gamma \supset_{P;R} \Delta$ is defined to be valid if every $(\Sigma \cup P; R)$-minimal model of Γ which satisfies $\bigwedge \Sigma$ satisfies $\bigvee \Delta$.

Definition 3.16 (the calculus PCC)
We define the propositional circumscription calculus PCC to have the deduction rules given in Figure 3.1.
In (C1) we call p the *principal constraint*. The minimized variable p and the fixed variable r are called the *principal formula* of (C3) and (C4), respectively.

The deduction rules (C1) and (C2) lead over to the calculus CPRC and CPC, respectively.
The intended meaning of the rules is given below.

(C1) If there is a P-minimal model of Γ which does not satisfy all constraints, then any formula is P-minimally entailed by Γ under those constraints.

(C2) The classical consequences of Γ are preserved under minimal entailment.

(C3) This rule allows to infer the validity of the sequent in the conclusion by distinguishing the different interpretations of a minimized variable.

(C4) This rule allows to infer the validity of the sequent in the conclusion by distinguishing the different interpretations of a fixed variable.

Compared to rule (C4) rule (C3) is not symmetric in its premises. The right premise of (C3) corresponds to Lemma 3.9 which says that a negated variable can be dragged into the list of minimal variables. Since there is no corresponding lemma for positive variables we can not put p into the antecedent. This is where we need the mechanism of constraints. So instead of putting p into the antecedent we define it to be a constraint for the left premise.

We now show that PCC is sound. Then we prove that the two branching rules (C3) and (C4) are invertible and can use this result to show completeness of PCC.

Theorem 3.17 (soundness)
If $\Sigma; \Gamma \supset_{P;R} \Delta$ is deducible in PCC, then it is valid.

Proof. We show our claim by induction on the number of applications of circumscription rules

- Let $\mathrm{S} := \Sigma, p; \Gamma \supset_P \Delta$ be deduced by (C1).

$$\frac{\mathsf{CPRC} \vdash \Gamma, \neg P \supset p}{\Sigma, p; \Gamma \supset_{P;\emptyset} \Delta}(\mathrm{C1})$$

  Then $\Gamma, \neg P \supset p$ is refutable in CPRC (1).
  Suppose S is not valid. Then there exists a model $M \vDash_{\Sigma,p,P;\emptyset} \Gamma$ such that $M \vDash \Sigma, p$ and $M \nvDash \bigvee \Delta$ (2). Since CPRC is sound we know from (1) that there exists a model $N \vDash \Gamma, \neg P$ with $p \notin N$. Thus

$$N \cap P = \emptyset \subseteq M \cap P \qquad N \cap \Sigma \subset \Sigma = M \cap \Sigma \qquad p \in M \setminus N$$

  which implies $N <_{\Sigma,p,P} M$. This contradicts to (2) since N is also a model of Γ. Hence S is valid.

- Let $\Sigma; \Gamma \supset_{P;R} \Delta$ be deduced by (C2). Then $\Sigma, \Gamma \supset \Delta$ is deducible in CPC and thus valid, that is $\bigvee \Delta$ is valid in every model of Γ that satisfies Σ.

- Let $\mathrm{S} := \Sigma; \Gamma \supset_{P,p;R} \Delta$ be deduced by (C3).

$$\frac{\Sigma, p; \Gamma \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg p \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P,p;R} \Delta}(\mathrm{C3})$$

  Suppose S is not valid. Then there exists a model $M \vDash_{\Sigma,P,p;R} \Gamma$ with $M \vDash \Sigma$ and $M \nvDash \bigvee \Delta$ (1). We distinguish two cases:

  1. $M \vDash p$: Then $M \vDash \Sigma, p$ which together with (1) contradicts to the induction hypothesis that $\Sigma, p; \Gamma \supset_{P;R} \Delta$ is valid.

2. $M \not\models p$: Then Corollary 3.8 yields $M \models_{\Sigma,P;R} \Gamma, \neg p$. With (1) this contradicts to the induction hypothesis that $\Sigma; \Gamma, \neg p \supset_{P;R} \Delta$ is valid.

Hence S is valid.

- Let $S := \Sigma; \Gamma \supset_{P;R,r} \Delta$ be deduced by (C4)

$$\frac{\Sigma; \Gamma, r \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg r \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P;R,r} \Delta}(\text{C4})$$

Suppose S is not valid. Then there exists a model $M \models_{\Sigma,P;R,r} \Gamma$ with $M \models \Sigma$ and $M \not\models \bigvee \Delta$ (1). We distinguish two cases:

1. $M \models r$: Then Lemma 3.10 yields $M \models_{\Sigma,P;R} \Gamma, r$. With (1) this contradicts to the induction hypothesis that $\Sigma; \Gamma, r \supset_{P;R} \Delta$ is valid.

2. $M \models \neg r$: Then Lemma 3.10 yields $M \models_{\Sigma,P;R} \Gamma, \neg r$. With (1) this contradicts to the induction hypothesis that $\Sigma; \Gamma, \neg r \supset_{P;R} \Delta$ is valid.

Hence S is valid.

□

To show completeness of PCC we use the fact that its branching rules are semantically invertible (see Definition 2.11).

Lemma 3.18 (semantically invertible rules)
The deduction rules (C3) and (C4) are semantically invertible.

Proof. Both rules are of the form

$$\frac{\Sigma_1; \Gamma_1 \supset_{P_1;R_1} \Delta \quad \Sigma_2; \Gamma_2 \supset_{P_2;R_2} \Delta}{\Sigma; \Gamma \supset_{P;R} \Delta}(\text{CX})$$

Consider the following sets of models.

$$\begin{aligned}\mathcal{M} &:= \{M : M \models_{\Sigma,P;R} \Gamma \text{ and } \Sigma \subset M\}\\ \mathcal{M}_1 &:= \{M_1 : M_1 \models_{\Sigma_1,P_1;R_1} \Gamma_1 \text{ and } \Sigma_1 \subset M_1\}\\ \mathcal{M}_2 &:= \{M_2 : M_2 \models_{\Sigma_2,P_2;R_2} \Gamma_2 \text{ and } \Sigma_2 \subset M_2\}\end{aligned}$$

We need to prove that if the conclusion is valid then so are the premises. The conclusion is valid if $M \models \bigvee \Delta$ for any $M \in \mathcal{M}$. By showing that $\mathcal{M}_1 \subseteq \mathcal{M}$ and $\mathcal{M}_2 \subseteq \mathcal{M}$ we then know that for any $M_1 \in \mathcal{M}_1$ and $M_2 \in \mathcal{M}_2$ we have $M_1 \models \bigvee \Delta$ and $M_2 \models \bigvee \Delta$ and thus know that the premises are valid, too.

- $\dfrac{\Sigma, p; \Gamma \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg p \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P,p;R} \Delta}$ (C3)

  Suppose $\Sigma; \Gamma \supset_{P,p;R} \Delta$ is valid and let

  $$\mathcal{M} := \{M : M \models_{\Sigma,P,p;R} \Gamma \text{ and } (\Sigma \cup p) \subset M\}$$

  – Left premise $(\Sigma, p; \Gamma \supset_{P;R} \Delta)$:

  Let $M_1 \models_{\Sigma,P,p;R} \Gamma$ and $\Sigma \cup \{p\} \subset M_1$. It is then trivial that $M_1 \in \mathcal{M}$.

  – Right premise $(\Sigma, \neg p; \Gamma \supset_{P;R} \Delta)$:

  Let $M_2 \models_{\Sigma,P;R} \Gamma, \neg p$ and $\Sigma \subset M_2$. Since $p \notin R$ we know by Lemma 3.9 that $M_2 \models_{\Sigma,P,p;R} \Gamma$ hence $M_2 \in \mathcal{M}$.

- $\dfrac{\Sigma; \Gamma, r \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg r \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P;R,r} \Delta}$ (C4)

  Suppose $\Sigma; \Gamma \supset_{P;R,r} \Delta$ is valid and let

  $$\mathcal{M} := \{M : M \models_{\Sigma,p;R,r} \Gamma \text{ and } \Sigma \subset M\}$$

  – Left premise $(\Sigma; \Gamma, r \supset_{P;R} \Delta)$:

  Let $M_2 \models_{\Sigma,P;R} \Gamma, r$ and $\Sigma \subset M_2$. Since $r \notin P$ we know by Lemma 3.11 that $M \models_{\Sigma,P;R,r} \Gamma$ hence $M_2 \in \mathcal{M}$.

  – Right premise $(\Sigma; \Gamma, \neg r \supset_{P;R} \Delta)$:

  Analogous to the previous case.

□

The other two circumscription rules are not semantically invertible. Take for example the sequent $p_1; p_1 \vee p_2 \supset_{p_2} \neg p_2$. This sequent is valid but $p_1 \vee p_2, \neg p_2 \supset p_1$ is not refutable and hence valid and $p_1, p_1 \vee p_2 \supset \neg p_2$ is not provable and hence invalid.

Theorem 3.19 (completeness)
If $\Sigma; \Gamma \supset_{P;R} \Delta$ is valid, then it is deducible in PCC.

Proof. Suppose that $\Sigma; \Gamma \supset_{P;R} \Delta$ is valid. Then we show our claim by induction on $|P| + |R|$.

- $|P| + |R| = 0$
  Suppose $\Sigma; \Gamma \supset_{\emptyset} \Delta$ $(*)$ is valid. We distinguish two cases:

  1. $\Gamma \nvdash \Sigma$
     Choose $p \in \Sigma$ such that $\Gamma \nvdash p$. Then $\Gamma \supset p$ is refutable and we can deduce $(*)$ with (C1)

2. $\Gamma \Vdash \Sigma$
   Then every model M of Γ fulfills Σ, thus $\Gamma \supset \Delta$ is valid and by monotonicity $\Sigma, \Gamma \supset \Delta$ is also valid and hence deducible in CPC. Using (C2) we can deduce $\Sigma; \Gamma \supset_\emptyset \Delta$

- $|P| + |R| > 0$
  Suppose $P \neq \emptyset$, $p \in P$ and let $P' := P \setminus \{p\}$. Since (C3) is semantically invertible we then know that $\Sigma, p; \Gamma \supset_{P';R} \Delta$ and $\Sigma; \Gamma, \neg p \supset_{P';R} \Delta$ are valid and deducible by induction hypothesis and can thus use (C3) to deduce $\Sigma; \Gamma \supset_{P',p;R} \Delta$.
  If P is empty then there exists $r \in R$. Let $R' := R \setminus \{r\}$. Since (C4) is semantically invertible we then know that $\Sigma; \Gamma, r \supset_{P';R} \Delta$ and $\Sigma; \Gamma, \neg r \supset_{P';R} \Delta$ are valid and deducible by induction hypothesis and can thus use (C4) to deduce $\Sigma; \Gamma \supset_{P;R',r} \Delta$.

□

Birds Can Fly

We close this chapter with the proof of the example given in the previous section showing that if our bird is not a penguin then it can fly.

Let $\Gamma := \{\mathsf{bird} \wedge \neg\mathsf{abn} \to \mathsf{fly}, \mathsf{penguin} \to \neg\mathsf{fly} \wedge \mathsf{bird}, \mathsf{bird}\}$, $P := \{\mathsf{abn}\}$. Then the proof that our bird can fly is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{fly}, \mathsf{bird} \supset \mathsf{abn}, \mathsf{penguin}}{\mathsf{fly}, \mathsf{penguin} \to (\neg\mathsf{fly} \wedge \mathsf{bird}), \mathsf{bird} \supset \mathsf{abn}}(\cdot \to \not\supset)}{\mathsf{CPRC} \vdash \Gamma \supset \mathsf{abn}}(\to \cdot \not\supset)}{\mathsf{abn}; \Gamma \supset_\emptyset \mathsf{fly}}(\text{C1}) \qquad \dfrac{\mathsf{CPC} \vdash \Gamma, \neg\mathsf{abn} \supset \mathsf{fly}}{; \Gamma, \neg\mathsf{abn} \supset_\emptyset \mathsf{fly}}(\text{C2})}{; \Gamma \supset_{\mathsf{abn}} \mathsf{fly}}(\text{C3})$$

The left branch of the proof covers the P-minimal models of Γ that contain abn. Since there are no such models we use the refutation calculus to show that abn does not follow from Γ and can then deduce the corresponding circumscription sequent with (C1).
The right branch of the proof covers the P-minimal models of Γ that do not contain abn. Assuming that our bird is not abnormal we can show in CPC that it can fly (we have omitted that part of the proof here). From that we can then deduce the corresponding circumscription sequent with (C2).

Chapter 4

Proof Search in Circumscription

This chapter covers automatic proving in propositional circumscription logic. We discuss two approaches.

First we have a look at the approach to compute $\mathrm{CIRC}(A, P, Q)$ and then use a classical prover to verify whether a circumscription sequent is valid. We will see that there is some additional information we can put into the circumscription theory in order to reduce its size. However, the computation of the circumscription theory itself is very time consuming since the size of the resulting theory is exponential in the number of minimized and varied variables. The approach is thus not very well suited for bigger problems.

Afterward we have a look at proof search in PCC. We first give two naive provers. The first of them prefers the invertible branching rules to the non-invertible non-branching rule, the second prover is doing it the other way round. Then we take a closer look at the second prover and introduce some general improvements that allow us to omit certain calls to the CPC and the CPRC prover. Afterward we develop use-check for the circumscription prover, present a modified calculus for proposition circumscription with implicit use-check and close the chapter with some experimental results.

4.1 Proving in Classical Logic

We have seen that it is possible to express circumscription using classical propositional logic (Definition 3.12 on page 60). To verify whether a formula is $(P; R)$-minimally entailed by a formula A is thus equivalent to verifying whether it is a logical consequence of $\mathrm{CIRC}(A, P, Q)$, where $Q := \mathrm{vars}(A) \setminus$

$(P \cup R)$.

The main advantage of this approach is obvious. There is no need to write an additional prover for circumscription and we can rely on our prover of classical propositional logic. All we need is to compute $\mathrm{CIRC}(A, P, Q)$.

However, this method has the disadvantage of the complexity of the circumscription theory $\mathrm{CIRC}(A, P, Q)$. The number of formulas in $\mathrm{CIRC}(A, P, Q)$ grows exponentially with the number of variables in P. Furthermore we are adding formulas of the form $C_{\underline{i}} \to D_{\underline{i}}$[1] to the antecedent. In proof search, each of the formulas is thus a potential branch. Moreover $C_{\underline{i}}$ is a conjunction of literals. When applying $(\to\supset)$ backwards $C_{\underline{i}}$ will be put into the succedent of the left premise. This means $|P|$ more potential branchings for each $B_{\underline{i}}$ we are adding.

Looking at the other part of the implication we see that the size of $D_{\underline{i}}$ depends on $\underline{i}$ and the number of varied formulas. We can enhance those formulas easily.

Enhancement 1

Since we are substituting in $D_{\underline{i}}$ the minimized and varied variables with $\top$ and $\bot$, we can use some basic boolean algebra to simplify those formulas. For example we can replace an occurrence of $p \to \bot$ with $\neg p$ or $p \wedge \bot$ with $\bot$ and so on. Like that we can reduce the complexity of $D_{\underline{i}}$ in certain circumstances enormously, for example if most of the variable are being minimized or varied. From an algorithmic point of view this is not much of an overhead since the simplification is done simultaneously to substituting the variables.

Enhancement 2

Regarding the validity of $B_{\underline{i}}$ we know that $D_{\underline{i}}$ must only be valid in a model M if $C_{\underline{i}}$ is valid in M, too. Since the interpretation of the minimized formulas is determined through $C_{\underline{i}}$, we can add $A[\underline{i}/P]$ to the conjunct in $D_{\underline{i}}$ without affecting the circumscription theory.

$$B_{\underline{i}}(A, P, Q) := \bigwedge p_k^{i_k} \to A[\underline{i}/P] \wedge \bigwedge_{\underline{j}<\underline{i}} \bigwedge_{\underline{k}\in\mathbb{B}^m} \neg A[\underline{j}/P, \underline{k}/Q]$$

The advantage here is that we put more detailed information into the circumscription theory. The best case is when we can use the method of point 1 to simplify $A[\underline{i}/P]$ to $\bot$ and thus reduce $B_{\underline{i}}$ to $\neg \bigwedge p_k^{i_k}$.

Enhancement 3

The third way to enhance $B_{\underline{i}}$ is similar to the second way but goes

[1] $C_{\underline{i}} := \bigwedge p_k^{i_k}$ and $D_{\underline{i}} := \bigwedge \neg A[\underline{j}/P, \underline{k}/Q]$

further in that the varied variables are also taken into account.

$$B_{\underline{i}}(A,P,Q) := \bigwedge p_k^{i_k} \rightarrow \left(\bigvee_{\underline{k}\in\mathbb{B}^m} A[\underline{i}/P, \underline{k}/Q] \right) \wedge \bigwedge_{\underline{j}<\underline{i}} \bigwedge_{\underline{k}\in\mathbb{B}^m} \neg A[\underline{j}/P, \underline{k}/Q]$$

The advantage is the same as in 2. But since also the varied variables are substitute with truth values, we might get a better simplification of the formula. However, we also have to add and minimize 2^m additional formulas.

These enhancements may certainly reduce the size of the circumscription theory. However, generally they will not prevent its the exponential growth.

4.2 Backward Proof Search

A more promising approach is that of backward proof search. As in the classical case we start with a very simple proof search algorithm that does nothing more than backward application of the rules. On an example we show that there are redundancies in this procedure and give a method how to detect them and how they can be avoided.

4.2.1 A Simple Proof Search Algorithm

The rules we have for circumscription sequents are either branching rules that have the nice property of being invertible or non-branching rules which are not invertible and will thus require backtracking. Algorithmically we can thus prefer branching rules to non-branching rules or vice versa.

When preferring branching to non-branching rules we are sure that no backtracking needs to be done as long as the set of minimized or fixed variables are not empty. However, if we use this approach for valid sequents then we end up with 2^n nodes, where n is the number of minimized and fixed variables. In each of those nodes we can then apply (C2) and (C1) backwards. This reflects exactly the approach of defining the circumscription theory in classical propositional logic. The pseudocode of this approach is given in Algorithm 3. As you see this is a very compact algorithm. Branching is done in lines 4 and 8. A minor modification has been done to rule (C1) (line 12). Instead of looping over the constraints we let the refutation function carry this out by adding the conjunction of the constraints to the succedent. This leads to situations where we allow (C1) to be applied backwards if the set of constraints is empty. However, this is still correct, since then the empty conjunction evaluates to $\top$ and thus the sequent will not be refutable.

Algorithm 3 Simple proof search for circumscription preferring branching rules to non-branching rules

```
1:  function circProvable-1(Σ; Γ ⊃_{P;R} Δ)
2:      if R ≠ ∅ then
3:          choose r ∈ R and let R' := R \ {r}
4:          success := circProvable-1(Σ; Γ, r ⊃_{P;R'} Δ) and
5:                     circProvable-1(Σ; Γ, ¬r ⊃_{P;R'} Δ)
6:      else if P ≠ ∅ then
7:          choose p ∈ P and let P' := P \ {p}
8:          success := circProvable-1(Σ, p; Γ ⊃_{P';R} Δ) and
9:                     circProvable-1(Σ; Γ, ¬p ⊃_{P';R} Δ)
10:     else
11:         success := cpcProvable(Σ, Γ ⊃ Δ) or
12:                    cpcRefutable(¬P, Γ ⊃ ⋀Σ)
13:     return success
```

When preferring non-branching to branching rules — the pseudocode is given in Algorithm 4 — we will need backtracking (lines 3 and 10). However, with this approach we can avoid branchings in the proof search if a non-branching rule succeeds for a sequent that has a non-empty set of minimized or fixed variables. The pseudocode also shows that we prefer rule (C4) to rule (C3). This is because (C1) can not be applied backwards if the set of fixed variables is not empty. Since our aim is to avoid branching in the proof search, we make certain that both non-branching rules can be applied as early as possible by having this preference. We also use the modified rule (C1) that lets the refutation function do the looping over the constraints (line 9).

Since circProvable-2 allows us to omit branches during proof search it seems as if it is to favor over circProvable-1. However, with additional improvements circProvable-1 can be quite efficient for certain problems. We have thus implemented both variants and point out the differences between them at the end of this chapter.

4.2.2 General Improvements

We now have a look at some improvements of the algorithm which are independent of the sequent that is to prove. These improvements help us to omit superfluous calls of cpcProvable and cpcRefutable and are mainly aimed at the algorithm circProvable-2. In the following we therefore only inspect circProvable-2 but will mention if a method is also applicable for circProvable-1.

Algorithm 4 Simple proof search for circumscription preferring non-branching rules to branching rules

```
1:  function circProvable-2(Σ; Γ ⊃_{P;R} Δ)
2:      success := cpcProvable(Σ, Γ ⊃ Δ)
3:      if not success then
4:          if R ≠ ∅ then
5:              choose r ∈ R and let R' := R \ {r}
6:              success := circProvable-2(Σ; Γ, r ⊃_{P;R'} Δ) and
7:                         circProvable-2(Σ; Γ, ¬r ⊃_{P;R'} Δ)
8:          else
9:              success := cpcRefutable(Γ, ¬P ⊃ ⋀Σ)
10:             if not success and P ≠ ∅ then
11:                 choose p ∈ P and let P' := P \ {p}
12:                 success := circProvable-2(Σ, p; Γ ⊃_{P';R} Δ) and
13:                            circProvable-2(Σ; Γ, ¬p ⊃_{P';R} Δ)
14:     return success
```

For the general improvements we analyze the order of the calls of cpcProvable and cpcRefutable, respectively. An example is given in Figure 4.1 and Figure 4.2. In both figures the left columns contain the sequents that are passed to circProvable-2. The right columns show the sequents that are passed to cpcProvable and cpcRefutable, respectively. Recursive calls are enumerated according to their position in the proof search tree.

Omitting CPC proofs

We first investigate the calls of cpcProvable. Suppose the algorithm reaches point 2, i.e. it does not fail for the first left branch. Then we know that the classical prover failed in the root node, that is $\nvdash \Gamma \supset \Delta$. Now the provability of the CPC sequent at position 2 needs only be verified if the prover previously failed in proving the CPC-sequent at position 1. If the prover succeeded for $\Gamma, r \supset \Delta$ (position 1), then $\Gamma, \neg r \supset \Delta$ (position 2) can not be provable since otherwise we could deduce that $\Gamma \supset \Delta$ is provable, which is, as we know at this point, not the case.

Omitting CPC refutations

While we use only one general method to omit calls of cpcProvable we use three obvious, easy to implement methods to omit calls of cpcRefutable.

1. As mentioned before, we use a modified version of the deduction rule (C1) that has the conjunction of all constraints in the succedent of the

$;\Gamma \supset_{p_1,p_2;r} \Delta$	$\Gamma \supset \Delta$
1 $;\Gamma, r \supset_{p_1,p_2} \Delta$	$\Gamma, r \supset \Delta$
1.1 $p_1;\Gamma, r \supset_{p_2} \Delta$	$p_1, \Gamma, r \supset \Delta$
1.1.1 $p_1, p_2;\Gamma, r \supset_{\emptyset} \Delta$	$p_1, p_2, \Gamma, r \supset \Delta$
1.1.2 $p_1;\Gamma, r, \neg p_2 \supset_{\emptyset} \Delta$	$p_1, \Gamma, r, \neg p_2 \supset \Delta$
1.2 $;\Gamma, r, \neg p_1 \supset_{p_2} \Delta$	$\Gamma, r, \neg p_1 \supset \Delta$
1.2.1 $p_2;\Gamma, r, \neg p_1 \supset_{\emptyset} \Delta$	$p_2, \Gamma, r, \neg p_1 \supset \Delta$
1.2.2 $;\Gamma, r, \neg p_1, \neg p_2 \supset_{\emptyset} \Delta$	$\Gamma, r, \neg p_1, \neg p_2, \supset \Delta$
2 $;\Gamma, \neg r \supset_{p_1,p_2} \Delta$	$\Gamma, \neg r \supset \Delta$
2.1 $p_1;\Gamma, \neg r \supset_{p_2} \Delta$	$p_1, \Gamma, \neg r \supset \Delta$
2.1.1 $p_1, p_2;\Gamma, \neg r \supset_{\emptyset} \Delta$	$p_1, p_2, \Gamma, \neg r \supset \Delta$
2.1.2 $p_1;\Gamma, \neg r, \neg p_2 \supset_{\emptyset} \Delta$	$p_1, \Gamma, \neg r, \neg p_2 \supset \Delta$
2.2 $;\Gamma, \neg r, \neg p_1 \supset_{p_2} \Delta$	$\Gamma, \neg r, \neg p_1 \supset \Delta$
2.2.1 $p_2;\Gamma, \neg r, \neg p_1 \supset_{\emptyset} \Delta$	$p_2, \Gamma, \neg r, \neg p_1 \supset \Delta$
2.2.2 $;\Gamma, \neg r, \neg p_1, \neg p_2 \supset_{\emptyset} \Delta$	$p_2, \Gamma, \neg r, \neg p_1, \neg p_2 \supset \Delta$

Figure 4.1: Invocations of CPCPROVABLE in circumscription prover

$;\Gamma \supset_{p_1,p_2,p_3;} \Delta$	$\Gamma, \neg p_1, \neg p_2, \neg p_3 \supset \top$
1 $p_1;\Gamma \supset_{p_2,p_3;} \Delta$	$\Gamma, \neg p_2, \neg p_3 \supset p_1$
1.1 $p_1, p_2;\Gamma \supset_{p_3;} \Delta$	$\Gamma, \neg p_3 \supset p_1 \wedge p_2$
1.1.1 $p_1, p_2, p_3;\Gamma \supset_{;} \Delta$	$\Gamma \supset p_1 \wedge p_2 \wedge p_3$
1.1.2 $p_1, p_2;\Gamma, \neg p_3 \supset_{;} \Delta$	$\Gamma, \neg p_3 \supset p_1 \wedge p_2$
1.2 $p_1;\Gamma, \neg p_2 \supset_{p_3;} \Delta$	$\Gamma, \neg p_2, \neg p_3 \supset p_1$
1.2.1 $p_1, p_3;\Gamma, \neg p_2 \supset_{;} \Delta$	$\Gamma, \neg p_2 \supset p_1 \wedge p_3$
1.2.2 $p_1;\Gamma, \neg p_2, \neg p_3 \supset_{;} \Delta$	$\Gamma, \neg p_2, \neg p_3 \supset p_1$
2 $;\Gamma, \neg p_1 \supset_{p_2,p_3;} \Delta$	$\Gamma, \neg p_1, \neg p_2, \neg p_3 \supset \top$
2.1 $p_2;\Gamma, \neg p_1 \supset_{p_3;} \Delta$	$\Gamma, \neg p_1, \neg p_3 \supset p_2$
2.1.1 $p_2, p_3;\Gamma, \neg p_1 \supset_{;} \Delta$	$\Gamma, \neg p_1 \supset p_2 \wedge p_3$
2.1.2 $p_2;\Gamma, \neg p_1, \neg p_3 \supset_{;} \Delta$	$\Gamma, \neg p_1, \neg p_3 \supset p_2$
2.2 $;\Gamma, \neg p_1, \neg p_2 \supset_{p_3;} \Delta$	$\Gamma, \neg p_1, \neg p_2, \neg p_3 \supset \top$
2.2.1 $p_3;\Gamma, \neg p_1, \neg p_2 \supset_{;} \Delta$	$\Gamma, \neg p_1, \neg p_2 \supset p_3$
2.2.2 $;\Gamma, \neg p_1, \neg p_2, \neg p_3 \supset_{;} \Delta$	$\Gamma, \neg p_1, \neg p_2, \neg p_3 \supset \top$

Figure 4.2: Invocations of CPCREFUTABLE in circumscription prover

premise. If there are no constraints, this leads to sequents having $\top$ as succedent. It is clear that such sequents are not refutable so we can omit trying to refute them.

2. We see that we have equivalent sequents at the positions 1, 1.2, 1.2.2, 2.1, 2.1.2 and 2.2.1. It is thus sufficient to verify refutability only at one of those position, for example at position 1.
   The positions in which this method can be applied are easy to identify since they are exactly those where the constraints of the circumscription sequent consists of a single variable.

3. We also see that the sequent in the call of the prover in the right branch is always equal to the sequent of its father node (positions 1.1.2 with father node 1.1 for example). It is therefore superfluous to verify refutability in the node of the right son if a backward-application of the rule (C1) was possible, and thus unsuccessful, in its father node.

The methods 1 and 2 are also applicable for circProvable-1.

A fourth method to omit calls of the refutation function relies on the following variant of monotonicity given in Proposition 1.19 on page 11.

Proposition 4.1 (monotonicity)
Let $\Gamma, \Gamma', \Delta, \Delta'$ be multisets of formulas. Then the following holds.

$$\text{If} \quad \models \Gamma \supset \bigwedge(\Delta \cup \Delta') \quad \text{then} \quad \models \Gamma, \Gamma' \supset \bigwedge \Delta.$$

In contrast to Proposition 1.19 we have a succedent with a conjunction of formulas instead of a disjunction of formulas. If we remove the formulas Δ' from that conjunction then the sequent will trivially still be valid. Together with monotonicity in the antecedent we obtain the proposition above.

The sequents we are trying to refute are all of the form $\Gamma, \neg R_0, R_1, \neg P_0 \supset \bigwedge P_1$, where (R_0, R_1) is a disjoint partition of the fixed variables R and (P_0, P_1) is a disjoint partition of the minimized variables P.
From the proposition above we thus know that if refuting $\Gamma, \neg R_0, R_1, \neg P_0 \supset \bigwedge P_1$ failed then refuting $\Gamma, \neg R_0, R_1, \neg P_0' \supset \bigwedge P_1'$ will also fail if $P_1' \subseteq P_1$. One such situation may for example be encountered at position 1.2.1 if refutation failed at position 1.1.1.

It is also clear how to identify the situations in which this fourth improvement can be applied. The variables in the conjunction of the succedent we try to refute are exactly the constraints of the corresponding circumscription sequent. It is thus sufficient to remember for each partition of R those sets of constraints for which backward application of (C1) failed. If we later encounter a circumscription sequent whose set of constraints is contained in one of the previously remembered sets of constraints, we will know that it will also fail for this sequent and can omit calling cpcRefutable.

This method also covers method 3 presented above and can, in contrast to method 3, also be applied in circProvable-1. For circProvable-2 it is therefore possible to drop method 3 in favor for it. However, compared to method 4 it is much easier to apply method 3 and dropping it is thus not advisable.

4.2.3 Use-Check

With the methods presented in the section above, we can mainly improve circProvable-2 with rather easy methods. The methods will however not be useful to prevent that circProvable-1 has to process a complete binary search tree for valid sequents. In this section we present a use-check method for propositional circumscription with which we can detect superfluous branchings. The method is not bound to a certain search strategy and thus applicable for both prover variants.

The underlying idea of the use-check method for propositional circumscription is similar to that of use-check in classical propositional logic. That is by analyzing the proof of one of the premises we may conclude the provability of the other premise. Since the calculus PCC includes the rule of CPC and CPRC, use-check can already be used for the subproofs that are done in these calculi. This leaves us with the question of how to avoid branchings for the rules (C3) and (C4)

$$\frac{\Sigma, p; \Gamma \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg p \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P,p;R} \Delta}(\text{C3}) \qquad \frac{\Sigma; \Gamma, r \supset_{P;R} \Delta \quad \Sigma; \Gamma, \neg r \supset_{P;R} \Delta}{\Sigma; \Gamma \supset_{P;R,r} \Delta}(\text{C4})$$

Let us have a look at (C4). An approach that is similar to use-check in CPC would be to check whether r is really needed in the proof of $\Sigma; \Gamma, r \supset_{P;R} \Delta$. However, if r is not needed then knowing the provability of $\Sigma; \Gamma \supset_{P;R} \Delta$ is of little use since circumscription is non-monotonic and we can then not deduce the provability of $\Sigma; \Gamma, \neg r \supset_{P;R} \Delta$ from it.
The approach we use for these two rules is to transform a proof of one premise into a proof of the other one.

Since we use the calculus CPC2 for proof search, we work in this section with a slightly modified version of PCC whose only difference to the original is that it is using labeled formulas and that (C1) and (C2) are rewritten as follows, where in (C1) the label n of $\overset{\text{n}}{p}$ is an arbitrary natural number.

$$\frac{\mathsf{CPRC} \vdash \overline{\Gamma}{\downarrow_{\mathbb{N}}}, \neg\overline{P}{\downarrow_{\mathbb{N}}} \supset p}{\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}}(\text{C1}) \qquad \frac{\mathsf{CPC2} \vdash \mathcal{D}; \overline{\Sigma}, \overline{\Gamma} \supset \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}}(\text{C2})$$

A labeled circumscription sequent $\overline{\Sigma};\overline{\Gamma} \supset_{\overline{P};\overline{P}} \overline{\Delta}$ is then defined to be valid if the circumscription sequent $\overline{\Sigma}{\downarrow_{\mathbb{N}}};\overline{\Gamma}{\downarrow_{\mathbb{N}}} \supset_{\overline{P}{\downarrow_{\mathbb{N}}};\overline{R}{\downarrow_{\mathbb{N}}}} \overline{\Delta}{\downarrow_{\mathbb{N}}}$ is valid. It should be clear that these modification do not change anything elementary in the calculus.

The following lemmas provide a basis for our proof transformations and consider deductions in CPC2 and CPRC.

Lemma 4.2 (switch side of irrelevant variables in CPC2-sequents)

1. If $\mathcal{D};\overline{\Gamma}, \overset{\text{n}}{p} \supset \overline{\Delta}$ is valid and $\text{n} \notin \mathcal{D}$ then $\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}, \overset{\text{n}}{p}$ is valid.
2. If $\mathcal{D};\overline{\Gamma}, \neg\overset{\text{n}}{p} \supset \overline{\Delta}$ is valid and $\text{n} \notin \mathcal{D}$ then $\mathcal{D};\overline{\Gamma}, \overset{\text{n}}{p} \supset \overline{\Delta}$ is valid.

Proof. The claim follows directly from the definition of the validity of a CPC2-sequent. □

Lemma 4.3 (switch side of variable in refutations)

1. Let $\mathcal{P}$ be a refutation of $\Gamma, p \supset \Delta$ and $P_\Gamma, p \supset P_\Delta$ be the anti-axiom of $\mathcal{P}$. If $p \notin P_\Gamma$ then there exists a refutation of $\Gamma \supset \Delta, p$ having the anti-axiom $P_\Gamma \supset P_\Delta, p$.
2. Let $\mathcal{P}$ be a refutation of $\Gamma \supset \Delta, p$ and $P_\Gamma \supset P_\Delta, p$ be the anti-axiom of $\mathcal{P}$. If $p \notin P_\Delta$ then there exists a refutation of $\Gamma, p \supset \Delta$ having the anti-axiom $P_\Gamma, p \supset P_\Delta$.

Proof.

1. Let $\mathcal{P}$ be a refutation of $\Gamma, p \supset \Delta$ having the anti-axiom $P_\Gamma, p \supset P_\Delta$. If $p \notin P_\Gamma$ then $P_\Gamma \supset P_\Delta, p$ is also an anti-axiom. Since p is atomic and therefore not deducible by any rule, it will appear in every antecedent of $\mathcal{P}$ and we obtain thus a refutation of $\Gamma \supset \Delta, p$ having the anti-axiom $P_\Gamma \supset P_\Delta, p$ if we remove p from every antecedent of $\mathcal{P}$ and add p to every succedent of $\mathcal{P}$.
2. Analogous.

□

Lemma 4.4 (remove negation on variables in refutations)
Let $\mathcal{P}$ be a refutation of $\Gamma, \neg p \supset \Delta$ and $P_\Gamma \supset P_\Delta, p$ be the anti-axiom of $\mathcal{P}$. If $p \notin P_\Delta$ then there exists a refutation of $\Gamma, p \supset \Delta$ having the anti-axiom $P_\Gamma, p \supset P_\Delta$.

Proof. Suppose $\mathcal{P}$ is a refutation of $\Gamma, \neg p \supset \Delta$ and $P_\Gamma \supset P_\Delta, p$ its anti-axiom with $p \notin P_\Delta$. We show our claim by induction on $\text{depth}(\mathcal{P})$.

- $\text{depth}(\mathcal{P}) = 1$. (Since $\neg p$ is not atomic, the depth of $\mathcal{P}$ is at least 1.)

  Then $\Gamma \supset \Delta, p$ is the anti-axiom of $\mathcal{P}$. Since $p \notin \Delta$ we know that $\Gamma, p \supset \Delta$ is an anti-axiom, too.

- $\text{depth}(\mathcal{P}) = n + 1$.

  We make a case distinction on the last rule of $\mathcal{P}$.

  – $\dfrac{\Gamma \supset \Delta, p}{\Gamma, \neg p \supset \Delta}(\neg\not\supset)$

    Then there exists a refutation of $\Gamma \supset \Delta, p$ that has $P_\Gamma \supset P_\Delta, p$ as its anti-axiom. According to Lemma 4.3 there exists thus a refutation of $\Gamma, p \supset \Delta$ that has $P_\Gamma, p \supset P_\Delta$ as its anti-axiom.

  – $\dfrac{\Gamma, \neg p \supset \Delta', A}{\Gamma, \neg p \supset \Delta', A \wedge B}(\not\supset\cdot\wedge)$ with $\Delta' := \Delta \setminus \{A \wedge B\}$.

    By induction hypothesis there exists a refutation of $\Gamma, p \supset \Delta', A$ having $P_\Gamma, p \supset P_\Delta$ as anti-axiom. With $(\not\supset \cdot\wedge)$ we can thus deduce $\Gamma, p \supset \Delta', A \wedge B$.

  The other cases are similar to the last case.

□

Lemma 4.5 (drop negative literals in refutation)
Let $\mathcal{P}$ be a refutation of $\Gamma, \neg p \supset \Delta$ and $P_\Gamma \supset P_\Delta, p$ be the anti-axiom of $\mathcal{P}$. Then there exists a refutation of $\Gamma \supset \Delta, p$ having the anti-axiom $P_\Gamma \supset P_\Delta, p$.

Proof. Suppose $\mathcal{P}$ is a refutation of $\Gamma, \neg p \supset \Delta$ and $P_\Gamma \supset P_\Delta, p$ its anti-axiom. We show our claim by induction on $\text{depth}(\mathcal{P})$.

- $\text{depth}(\mathcal{P}) = 1$. (Since $\neg p$ is not atomic, the depth of $\mathcal{P}$ is at least 1.)

  Then $\Gamma \supset \Delta, p$ is the anti-axiom of $\mathcal{P}$ and the refutation we are looking for.

- $\text{depth}(\mathcal{P}) = n + 1$.

  We make a case distinction on the last rule of $\mathcal{P}$.

  – $\dfrac{\Gamma \supset \Delta, p}{\Gamma, \neg p \supset \Delta}(\neg\not\supset)$

    Then $\Gamma \supset \Delta, p$ is trivially refutable with the required anti-axiom.

  – $\dfrac{\Gamma, \neg p \supset \Delta', A}{\Gamma, \neg p \supset \Delta', A \wedge B}(\not\supset\cdot\wedge)$ with $\Delta' := \Delta \setminus \{A \wedge B\}$.

    By induction hypothesis there exists a refutation of $\Gamma \supset \Delta', A, p$ having $P_\Gamma \supset P_\Delta, p$ as anti-axiom. With $(\not\supset \cdot\wedge)$ we can thus deduce $\Gamma \supset \Delta', A \wedge B, p$.

The other cases are similar to the last case.

□

We can remove a proposition variable from a refutation if it is part of the refuted sequent.

Lemma 4.6 (monotonicity in refutations)
Let $\mathcal{P}$ be a refutation of $\Gamma, p \supset \Delta$ ($\Gamma \supset \Delta, p$) and $P_\Gamma, p \supset P_\Delta$ ($P_\Gamma \supset P_\Delta, p$) be the anti-axiom of $\mathcal{P}$. Then there exists a refutation of $\Gamma \supset \Delta$ having the anti-axiom $P_\Gamma \supset P_\Delta$.

Proof. We know by monotonicity that $P_\Gamma \supset P_\Delta$ is refutable. We thus obtain a refutation of $\Gamma \supset \Delta$ if we remove one occurrence of p from the antecedent (succedent) of every sequent of $\mathcal{P}$. □

We can add a proposition variable to a refutation if it does not occur on the other side of its anti-axiom.

Lemma 4.7 (add variables to refutations)
Let $\mathcal{P}$ be a refutation of $\Gamma \supset \Delta$ and $P_\Gamma \supset P_\Delta$ be the anti-axiom of $\mathcal{P}$.

1. If $p \notin P_\Gamma$ then there exists a refutation of $\Gamma \supset \Delta, p$ which has $P_\Gamma \supset P_\Delta, p$ as its anti-axiom.
2. If $p \notin P_\Delta$ then there exists a refutation of $\Gamma, p \supset \Delta$ which has $P_\Gamma, p \supset P_\Delta$ as its anti-axiom.

Proof.

1. Since $p \notin P_\Gamma$ we know that $P_\Gamma \supset P_\Delta, p$ is an anti-axiom. We thus obtain a refutation of $\Gamma \supset \Delta, p$ with that anti-axiom if we add p to every succedent of the refutation of $\Gamma \supset \Delta$
2. Analogous.

□

The following theorem gives us the conditions that must be fulfilled in the proof of the left premise of (C3) to omit proving the right premise.

Theorem 4.8 (omit right premise of (C3)**)**
If there exists a proof $\mathcal{P}$ of $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ such that

1. $\text{n} \notin \mathcal{D}$ for every $\mathsf{CPC2}$-sequent $\mathcal{D}; \overline{\Gamma'} \supset \overline{\Delta}$ of a (C2)-rule application in $\mathcal{P}$,
2. p does not appear in the antecedent of any anti-axiom of $\mathcal{P}$,

3. there is no step of the form $\dfrac{\mathsf{CPRC} \vdash \overline{\Gamma'}{\downarrow_{\mathbb{N}}}, \neg\overline{P'}{\downarrow_{\mathbb{N}}} \supset p}{\overline{\Sigma'}, \overset{\text{n}}{p}; \overline{\Gamma'} \supset_{\overline{P'}} \overline{\Delta'}}$(C1) in $\mathcal{P}$,

then $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ is provable, too.

Proof. Let $\mathcal{P}$ be a proof of $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ that has the properties as given above. We show our claim by induction on the number c of circumscription rules in $\mathcal{P}$.

- $c = 1$:

  Then the last rule of $\mathcal{P}$ was either (C1) or (C2):

  – $\dfrac{\mathsf{CPRC} \vdash \Gamma, \neg P \supset s}{\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P}} \overline{\Delta}}$(C1) with $\overset{\text{m}}{s} \in \overline{\Sigma}$ and $\overset{\text{n}}{p} \notin \overline{\Sigma}$.

  Let $P_\Gamma \supset P_\Delta$ be the anti-axiom of the refutation of $\Gamma, \neg P \supset s$.

  Since $p \notin P_\Gamma$ we know by Corollary 4.7 that $\Gamma, \neg P \supset s, p$ is refutable. With $(\neg \not\supset)$ and (C1) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P}} \overline{\Delta}$.

  – $\dfrac{\mathsf{CPC2} \vdash \mathcal{D}; \overline{\Sigma}, \overset{\text{n}}{p}, \overline{\Gamma} \supset \overline{\Delta}}{\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}}$(C2) with $\text{n} \notin \mathcal{D}$.

  Since $\text{n} \notin \mathcal{D}$ we know by Lemma 4.2 that $\mathcal{D}; \overline{\Sigma}, \overline{\Gamma} \supset \overline{\Delta}, \overset{\text{n}}{p}$ is valid. With $(\neg \supset)$ and (C2) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P};\overline{R}} \overline{\Delta}$.

- $c > 1$:

  Then the last rule of $\mathcal{P}$ was either (C3) or (C4):

  – $\dfrac{\overline{\Sigma}, \overset{\text{n}}{p}, \overset{\text{m}}{q}; \overline{\Gamma} \supset_{\overline{P'};\overline{R}} \overline{\Delta} \quad \overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma}, \overset{\text{m}}{\neg q} \supset_{\overline{P'};\overline{R}} \overline{\Delta}}{\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P'}, \overset{\text{m}}{q};\overline{R}} \overline{\Delta}}$(C3) with $\overline{P} = \overline{P'}, \overset{\text{m}}{q}$.

  Then $\overline{\Sigma}, \overset{\text{m}}{q}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P'},\overline{R}} \overline{\Delta}$ and $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{m}}{\neg q}, \overset{\text{n}}{\neg p} \supset_{\overline{P'},\overline{R}} \overline{\Delta}$ are provable by induction hypothesis. With (C3) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P'}, \overset{\text{m}}{q};\overline{R}} \overline{\Delta}$.

  – $\dfrac{\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma}, \overset{\text{m}}{r} \supset_{\overline{P};\overline{R'}} \overline{\Delta} \quad \overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma}, \overset{\text{m}}{\neg r} \supset_{\overline{P};\overline{R'}} \overline{\Delta}}{\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R'}, \overset{\text{m}}{r}} \overline{\Delta}}$(C4) with $\overline{R} = \overline{R'}, \overset{\text{m}}{r}$.

  Then $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p}, \overset{\text{m}}{r} \supset_{\overline{P};\overline{R'}} \overline{\Delta}$ and $\overline{\Sigma}; \overset{\text{n}}{\neg p}, \overset{\text{m}}{\neg r}, \overline{\Gamma} \supset_{\overline{P};\overline{R'}} \overline{\Delta}$ are provable by induction hypothesis. With (C4) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P};\overline{R'}, \overset{\text{m}}{r}} \overline{\Delta}$.

□

The following theorem gives us the conditions that must be fulfilled in the proof of the right premise of (C3) to omit proving the left premise.

Theorem 4.9 (omit left premise of (C3)**)**
If there exists a proof $\mathcal{P}$ of $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ such that $\text{n} \notin \mathcal{D}$ for every $\mathsf{CPC2}$-sequent $\mathcal{D}; \overline{\Gamma}' \supset \overline{\Delta}'$ of a (C2)-rule application in $\mathcal{P}$, then $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ is provable, too.

Proof. Let $\mathcal{P}$ be a proof of $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ that has the properties as given above. We show our claim by induction on the number c of circumscription rules in $\mathcal{P}$.

- $c = 1$:

  Then the last rule of $\mathcal{P}$ was either (C1) or (C2):

  – $\dfrac{\mathsf{CPRC} \vdash \Gamma, \neg p, \neg P \supset s}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P}} \overline{\Delta}}$(C1) with $\overset{\text{m}}{s} \in \overline{\Sigma}$.

  From the premise we know by Lemma 4.5 and Corollary 4.6 that $\Gamma, \neg P \supset p$ is refutable. With (C1) we can infer $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P}} \overline{\Delta}$.

  – $\dfrac{\mathsf{CPC2} \vdash \mathcal{D}; \overline{\Sigma}, \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P};\overline{R}} \overline{\Delta}}$(C2) with $\text{n} \notin \mathcal{D}$.

  Since $\text{n} \notin \mathcal{D}$ we know by Lemma 4.2 that $\mathcal{D}; \overline{\Sigma}, \overline{\Gamma}, \overset{\text{n}}{p} \supset \overline{\Delta}$ is provable. With (C2) we can infer $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$.

- $c > 1$:

  Then the last rule of $\mathcal{P}$ was either (C3) or (C4):

  – $\dfrac{\overline{\Sigma}, \overset{\text{m}}{q}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P}';\overline{R}} \overline{\Delta} \quad \overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p}, \overset{\text{m}}{\neg q} \supset_{\overline{P}';\overline{R}} \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P}', \overset{\text{m}}{q};\overline{R}} \overline{\Delta}}$(C3) with $\overline{P} = \overline{P}', \overset{\text{m}}{q}$.

  Then $\overline{\Sigma}, \overset{\text{n}}{p}, \overset{\text{m}}{q}; \overline{\Gamma} \supset_{\overline{P}',\overline{R}} \overline{\Delta}$ and $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma}, \overset{\text{m}}{\neg q} \supset_{\overline{P}',\overline{R}} \overline{\Delta}$ are provable by induction hypothesis. With (C3) we can infer $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P}', \overset{\text{m}}{q};\overline{R}} \overline{\Delta}$.

  – $\dfrac{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p}, \overset{\text{m}}{r} \supset_{\overline{P};\overline{R}'} \overline{\Delta} \quad \Sigma; \overline{\Gamma}, \overset{\text{n}}{\neg p}, \overset{\text{m}}{\neg r} \supset_{\overline{P};\overline{R}'} \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{\overline{P};\overline{R}', \overset{\text{m}}{r}} \overline{\Delta}}$(C4) with $\overline{R} = \overline{R}', \overset{\text{m}}{r}$.

  Then $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma}, \overset{\text{m}}{r} \supset_{\overline{P};\overline{R}'} \overline{\Delta}$ and $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma}, \overset{\text{m}}{\neg r} \supset_{\overline{P};\overline{R}'} \overline{\Delta}$ are provable by induction hypothesis. With (C4) we can infer $\overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}', \overset{\text{m}}{r}} \overline{\Delta}$.

□

The following theorem gives us the conditions that must be fulfilled in the proof of the left premise of (C4) to omit proving the right premise.

Theorem 4.10 (omit right premise of (C4)**)**
If there exists a proof $\mathcal{P}$ of $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ such that

1. $\text{n} \notin \mathcal{D}$ for every $\mathsf{CPC2}$-sequent $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ of a (C2)-rule application in $\mathcal{P}$,
2. r occurs only once in each antecedent of the anti-axioms of $\mathcal{P}$,

then $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ is provable, too.

Proof. Let $\mathcal{P}$ be a proof of $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ that has the properties as given above. We show our claim by induction on the number c of circumscription rules in $\mathcal{P}$.

- $c = 1$:

  Then the last rule of $\mathcal{P}$ was either (C1) or (C2):

  – $\dfrac{\mathsf{CPRC} \vdash \Gamma, r, \neg P \supset s}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P}} \overline{\Delta}}$(C1) with $\overset{\text{m}}{s} \in \overline{\Sigma}$.

  Since r occurs only once in the antecedent of the anti-axiom of the refutation of $\Gamma, r, \neg P \supset s$, we know by Lemma 4.3 that $\Gamma, \neg P \supset s, r$ is refutable. With $(\neg\, \not\supset)$ and (C1) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r} \supset_{\overline{P}} \overline{\Delta}$.

  – $\dfrac{\mathsf{CPC2} \vdash \mathcal{D}; \overline{\Sigma}, \overline{\Gamma}, \overset{\text{n}}{r} \supset \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P};\overline{R}} \overline{\Delta}}$(C2) with $\text{n} \notin \mathcal{D}$.

  Since $\text{n} \notin \mathcal{D}$ we know by Lemma 4.2 that $\mathcal{D}; \overline{\Sigma}, \overline{\Gamma} \supset \overline{\Delta}, \overset{\text{n}}{r}$ is provable. With $(\neg \supset)$ and (C2) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r} \supset_{\overline{P};\overline{R}} \overline{\Delta}$.

- $c > 1$:

  Then the last rule of $\mathcal{P}$ was either (C3) or (C4):

  – $\dfrac{\overline{\Sigma}, \overset{\text{m}}{p}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P}';\overline{R}} \overline{\Delta} \quad \overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r}, \overset{\text{m}}{\neg p} \supset_{\overline{P}';\overline{R}} \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P}', \overset{\text{m}}{p};\overline{R}} \overline{\Delta}}$(C3) with $\overline{P} = \overline{P}', \overset{\text{m}}{p}$.

  Then $\overline{\Sigma}, \overset{\text{m}}{p}; \overline{\Gamma}, \overset{\text{n}}{\neg r} \supset_{\overline{P}',\overline{R}} \overline{\Delta}$ and $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r}, \overset{\text{m}}{\neg p} \supset_{\overline{P}',\overline{R}} \overline{\Delta}$ are provable by induction hypothesis. With (C3) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r} \supset_{\overline{P}', \overset{\text{m}}{p};\overline{R}} \overline{\Delta}$.

  – $\dfrac{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r}, \overset{\text{m}}{q} \supset_{\overline{P};\overline{R}'} \overline{\Delta} \quad \overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r}, \overset{\text{m}}{\neg q} \supset_{\overline{P};\overline{R}'} \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P};\overline{R}', \overset{\text{m}}{q}} \overline{\Delta}}$(C4) with $\overline{R} = \overline{R}', \overset{\text{m}}{q}$.

Then $\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r, {}^{\mathrm{m}} q \supset_{\overline{P};\overline{R}'} \overline{\Delta}$ and $\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r, \neg^{\mathrm{m}} q \supset_{\overline{P};\overline{R}'} \overline{\Delta}$ are provable by induction hypothesis. With (C4) we can infer $\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r \supset_{\overline{P};\overline{R}', {}^{\mathrm{m}} q} \overline{\Delta}$.

□

The following theorem gives us the conditions that must be fulfilled in the proof of the right premise of (C4) to omit proving the left premise.

Theorem 4.11 (omit left premise of (C4)**)**
If there exists a proof $\mathcal{P}$ of $\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r \supset_{\overline{P};\overline{R}} \overline{\Delta}$ such that

1. $\mathrm{n} \notin \mathcal{D}$ for every $\mathsf{CPC2}$-sequent $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ of a (C2)-rule application in $\mathcal{P}$,
2. r occurs only once in each succedent of the anti-axioms of $\mathcal{P}$,

then $\overline{\Sigma}; \overline{\Gamma}, {}^{\mathrm{n}} r \supset_{\overline{P};\overline{R}} \overline{\Delta}$ is provable, too.

Proof. Let $\mathcal{P}$ be a proof of $\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r \supset_{\overline{P};\overline{R}} \overline{\Delta}$ that has the properties as given above. We show our claim by induction on the number c of circumscription rules in $\mathcal{P}$.

- $c = 1$:

  Then the last rule of $\mathcal{P}$ was either (C1) or (C2):

  – $\dfrac{\mathsf{CPRC} \vdash \Gamma, \neg r, \neg P \supset s}{\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r \supset_{\overline{P}} \overline{\Delta}}$(C1) with ${}^{\mathrm{m}} s \in \Sigma$.

  Since r occurs only once in the succedent of the axiom of the refutation of $\Gamma, \neg r, \neg P \supset s$, we know by Lemma 4.4 that $\Gamma, r, \neg P \supset s$ is refutable. With (C1) we can infer $\overline{\Sigma}; \overline{\Gamma}, {}^{\mathrm{n}} r \supset_{\overline{P}} \overline{\Delta}$.

  – $\dfrac{\mathsf{CPC} \vdash \mathcal{D}; \overline{\Sigma}, \overline{\Gamma}, \neg^{\mathrm{n}} r \supset \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r \supset_{\overline{P};\overline{R}} \overline{\Delta}}$(C2) with $\mathrm{n} \notin \mathcal{D}$

  Since $\mathrm{n} \notin \mathcal{D}$ we know by Lemma 4.2 that $\mathcal{D}; \overline{\Sigma}, \overline{\Gamma}, {}^{\mathrm{n}} r \supset \overline{\Delta}$ is provable. With (C2) we can infer $\overline{\Sigma}; \overline{\Gamma}, {}^{\mathrm{n}} r \supset_{\overline{P};\overline{R}} \overline{\Delta}$.

- $c > 1$:

  Then the last rule of $\mathcal{P}$ was either (C3) or (C4):

  – $\dfrac{\overline{\Sigma}, {}^{\mathrm{m}} p; \overline{\Gamma}, \neg^{\mathrm{n}} r \supset_{\overline{P}';\overline{R}} \overline{\Delta} \quad \overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r, \neg^{\mathrm{m}} p \supset_{\overline{P}';\overline{R}} \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \neg^{\mathrm{n}} r \supset_{\overline{P}', {}^{\mathrm{m}} p;\overline{R}} \overline{\Delta}}$(C3) with $\overline{P} = \overline{P}', {}^{\mathrm{m}} p$.

Then $\overline{\Sigma}, \overset{\text{m}}{p}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P}',\overline{R}} \overline{\Delta}$ and $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r}, \overset{\text{m}}{\neg p} \supset_{\overline{P}',\overline{R}} \overline{\Delta}$ are provable by induction hypothesis. With (C3) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P}',\overset{\text{m}}{p};\overline{R}} \overline{\Delta}$.

– $\dfrac{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r}, \overset{\text{m}}{q} \supset_{\overline{P};\overline{R}'} \overline{\Delta} \quad \overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r}, \overset{\text{m}}{\neg q} \supset_{\overline{P};\overline{R}'} \overline{\Delta}}{\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r} \supset_{\overline{P};\overline{R}',\overset{\text{m}}{q}} \overline{\Delta}}$(C4) with $\overline{R} = \overline{R}', \overset{\text{m}}{q}$.

Then $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r}, \overset{\text{m}}{q} \supset_{\overline{P};\overline{R}'} \overline{\Delta}$ and $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r}, \overset{\text{m}}{\neg q} \supset_{\overline{P};\overline{R}'} \overline{\Delta}$ are provable by induction hypothesis. With (C4) we can infer $\overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P};\overline{R}',\overset{\text{m}}{q}} \overline{\Delta}$.

□

A Proof Search Algorithm With Use-Check

The theorems that allow us to omit proving one of the premises of (C3) or (C4) rely on three properties of the proof of the already proven premise:

1. The use-sets of the (C2) rules.
2. The minimized or fixed variables that occur in the anti-axioms.
3. The principal constraints of the (C1) rules.

To apply use-check for (C3) and (C4) it is therefore sufficient to know the set of anti-axioms, the use-sets of the (C2)-rule applications and the set of principal constraints of the (C1)-rules. When using labeled formulas the latter information can also be stored in the use-set that is computed from the (C2)-rules (see below).

The theorems rely on transforming the proof of one premise into the proof of the other premise. These proof transformations have two important properties.

1. They do not change the use-sets of the CPC2-sequents.
2. The changes done to the anti-axioms are of a very specific nature. We move one propositional variable from one side of the sequent to the other. Since the propositional variable that we move in the anti-axioms corresponds to the principal constraint of the circumscription rule in question, the anti-axioms of the transformed proof will not add additional information that will be of relevance for use-check in another circumscription rule.

Because of these two properties additional information from the transformed proof need not be taken into account for use-check.

Based on the theorems above we can now write down a proof-search algorithm doing use-check. It is given in Algorithm 5. Besides the circum-

scription sequent the functions take two additional arguments: A use-set $\mathcal{D}$ holding the united use-sets of the (C2)-rule applications and the labels of the principal constraints of the (C1)-rules, and a set of sequents $\mathcal{S}$ holding the anti-axioms.

In the algorithm we call the functions CPC2PROVABLE and CPCREFUTABLE whose pseudo-code is not given. We assume that the use-set of the CPC2-proof is returned through the additionally given argument $\mathcal{D}$. Furthermore we assume that CPCREFUTABLE stores the anti-axiom in the additionally given argument S. In all functions $\mathcal{D}$, $\mathcal{S}$ and S are passed by reference while the other arguments are passed by value.
Use-check is done in lines 19 (Theorem 4.8), 22 (Theorem 4.9), 33 (Theorem 4.10) and 36 (Theorem 4.11).

Example

To discuss some aspects of the algorithm we look at a proof of the sequent $r \overset{3}{\rightarrow} q, q \wedge \overset{3}{r} \rightarrow s, \neg q \overset{3}{\rightarrow} s \supset_{\overline{P};\overset{1}{\mathbf{q}},\overset{2}{r}} \overset{3}{s}$. A part of a possible proof that could be found by the algorithm is given in Figure 4.3. In the figure we have omitted the set of anti-axioms since we do not make use of refutations in this example.

We see that in the first backward application of the rule (C4) use-check is successful because of the label of $\overset{1}{q}$ is not element of the use-set[2]. Looking at the CPC2-axioms we see that in the axiom $(\{4,7\}\,; \overset{4}{\mathbf{q}}, r \overset{3}{\vee} s, \overset{1}{q}, \overset{2}{r} \supset \overset{3}{s}, \overset{7}{\mathbf{q}})$ we had the choice between $\overset{4}{q}$ or $\overset{1}{q}$ to form the identity axiom. Choosing $\overset{4}{q}$ lead to a successful use-check for (C4). We could as well have chosen $\overset{1}{q}$ to form the axiom in which case use-check would have failed. The choice of the principal formula in the CPC2-axioms can therefore be crucial for use-check. To have the highest chance that use-check in the circumscription rules succeeds we should therefore choose in the CPC2-axioms those principal formulas which do not originate from the set of minimized or fixed variables.

To maximize the benefit of use-check, all formulas in the circumscription sequent should be labeled with a different value. The CPC2 prover should then also use fresh labels for formulas resulting from branching rules. Like this no spurious dependencies are introduced for use-check.

[2]since the set of anti-axioms is empty, the second condition of Theorem 4.11 is also fulfilled

Algorithm 5 Proof search with use-check for circumscription

function circProvableUC($\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$)
  success := cpc2Provable($\mathcal{D}; \overline{\Sigma}, \overline{\Gamma} \supset \overline{\Delta}$)
  if not success **then**
    if $\overline{R} \neq \emptyset$ **then**
      success = c4($\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$)
    else
      for all $\overset{\text{n}}{s} \in \overline{\Sigma}$ **do**
        success := cpcRefutable($\mathrm{S}; \overline{\Gamma}{\downarrow_{\mathbb{N}}}, \neg\overline{P}{\downarrow_{\mathbb{N}}} \supset s$)
        if success **then**
          $\mathcal{D} := \{n\}$ and $\mathcal{S} := \{\mathrm{S}\}$
          break
      if not success **and** $P \neq \emptyset$ **then**
        success := c3($\mathcal{S}; \mathcal{D}; \Sigma; \Gamma \supset_{P;R} \Delta$)
  return success

function c3($\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$)
  choose $\overset{\text{n}}{p} \in \overline{P}$ and let $\overline{P}' := \overline{P} \setminus \left\{\overset{\text{n}}{p}\right\}$
  success := circProvableUC($\mathcal{S}; \mathcal{D}; \overline{\Sigma}, \overset{\text{n}}{p}; \overline{\Gamma} \supset_{\overline{P}';\overline{R}} \overline{\Delta}$)
  if success **then**
    if $\mathrm{n} \in \mathcal{D}$ **or** $p \in \Gamma'$ for a $\Gamma' \supset \Delta' \in \mathcal{S}$ **then**
      success := circProvableUC($\mathcal{S}'; \mathcal{D}'; \overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg p} \supset_{P';R} \Delta$)
      if success **then**
        if $\mathrm{n} \in \mathcal{D}'$ **then**
          $\mathcal{D} := \mathcal{D} \cup \mathcal{D}'$ and $\mathcal{S} := \mathcal{S} \cup \mathcal{S}'$
        else
          $\mathcal{D} := \mathcal{D}'$ and $\mathcal{S} := \mathcal{S}'$
      else
        $\mathcal{D} := \emptyset$ and $\mathcal{S} := \emptyset$
  return success

function c4($\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$)
  choose $\overset{\text{n}}{r} \in \overline{R}$ and let $\overline{R}' := \overline{R} \setminus \left\{\overset{\text{n}}{r}\right\}$
  success := circProvableUC($\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{r} \supset_{\overline{P};\overline{R}'} \overline{\Delta}$)
  if success **then**
    if $\mathrm{n} \in \mathcal{D}$ **or** $\mathrm{count}(\Gamma', r) > 1$ for a $\Gamma' \supset \Delta' \in \mathcal{S}$ **then**
      success := circProvableUC($\mathcal{S}'; \mathcal{D}'; \overline{\Sigma}; \overline{\Gamma}, \overset{\text{n}}{\neg r} \supset_{\overline{P};\overline{R}'} \overline{\Delta}$)
      if success **then**
        if $\mathrm{n} \in \mathcal{D}'$ **or** $\mathrm{count}(\Delta', r) > 1$ for a $\Gamma' \supset \Delta' \in \mathcal{S}'$ **then**
          $\mathcal{D} := \mathcal{D} \cup \mathcal{D}'$ and $\mathcal{S} := \mathcal{S} \cup \mathcal{S}'$
        else
          $\mathcal{D} := \mathcal{D}'$ and $\mathcal{S} := \mathcal{S}'$
      else
        $\mathcal{D} := \emptyset$ and $\mathcal{S} := \emptyset$
  return success

$$\dfrac{\dfrac{}{\{2,4\}\,;q\wedge \overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q},\overset{2}{\mathbf{r}}\supset\overset{3}{s},\overset{4}{\mathbf{r}}}\,(\mathsf{id}) \qquad \dfrac{\dfrac{}{\{4,7\}\,;\overset{4}{\mathbf{q}},r\overset{3}{\vee}s,\overset{1}{q},\overset{2}{r}\supset\overset{3}{s},\overset{7}{\mathbf{q}}}\,(\mathsf{id}) \quad \dfrac{}{\{2,7\}\,;\overset{4}{q},r\overset{3}{\vee}s,\overset{1}{q},\overset{2}{\mathbf{r}}\supset\overset{3}{s},\overset{7}{\mathbf{r}}}\,(\mathsf{id})}{\{2,4,6\}\,;\overset{4}{q},r\overset{3}{\vee}s,\overset{1}{q},\overset{2}{r}\supset\overset{3}{s},\mathbf{q}\overset{6}{\wedge}\mathbf{r}}\,(\supset\wedge) \qquad \dfrac{}{\{3,6\}\,;\overset{4}{q},r\overset{3}{\vee}s,\overset{6}{\mathbf{s}},\overset{1}{q},\overset{2}{r}\supset\overset{3}{\mathbf{s}}}\,(\mathsf{id})}{\{2,3,4\}\,;\overset{4}{q},\mathbf{q}\wedge\overset{3}{\mathbf{r}}\to\mathbf{s},r\overset{3}{\vee}s,\overset{1}{q},\overset{2}{r}\supset\overset{3}{s}}\,(\to\supset)}{\{2,3\}\,;\mathbf{r}\overset{3}{\to}\mathbf{q},q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q},\overset{2}{r}\supset\overset{3}{s}}\,(\to\supset)$$

(1)

$$\dfrac{\dfrac{}{\{2,4\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,\overset{4}{\mathbf{r}},\overset{1}{q}\supset\overset{3}{s},\overset{2}{\mathbf{r}}}\,(\mathsf{id}) \qquad \dfrac{}{\{3,4\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,\overset{4}{\mathbf{s}},\overset{1}{q}\supset\overset{3}{\mathbf{s}},\overset{2}{r}}\,(\mathsf{id})}{\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,\mathbf{r}\overset{3}{\vee}\mathbf{s},\overset{1}{q}\supset\overset{3}{s},\overset{2}{r}}\,(\vee\supset)}{\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q},\neg\overset{2}{\mathbf{r}}\supset\overset{3}{s}}\,(\neg\supset)$$

(2)

$$\dfrac{\dfrac{\text{(1)}}{\mathsf{CPC}\vdash\{2,3\}\,;\mathbf{r}\overset{3}{\to}\mathbf{q},q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q},\overset{2}{r}\supset\overset{3}{s}}\,(\mathrm{C2}) \qquad \dfrac{\text{(2)}}{\mathsf{CPC}\vdash\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q},\neg\overset{2}{\mathbf{r}}\supset\overset{3}{s}}\,(\mathrm{C2})}{\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q},\neg\overset{2}{r}\supset_{\overline{P}}\overset{3}{s}}\,(\mathrm{C4})}{\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q}\supset_{\overline{P};\overset{2}{\mathbf{r}}}\overset{3}{s}}$$

(3)

$$\dfrac{\dfrac{\text{(3)}}{\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s,\overset{1}{q}\supset_{\overline{P};\overset{2}{\mathbf{r}}}\overset{3}{s}} \qquad \dfrac{\text{provable (use-check)}}{\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,\neg q\overset{3}{\to}s,\neg\overset{1}{q}\supset_{\overline{P};\overset{2}{\mathbf{r}}}\overset{3}{s}}}{\{2,3\}\,;r\overset{3}{\to}q,q\wedge\overset{3}{r}\to s,r\overset{3}{\vee}s\supset_{\overline{P};\overset{1}{\mathbf{q}},\overset{2}{r}}\overset{3}{s}}\,(\mathrm{C4})$$

Figure 4.3: Use-check in circumscription

The Order of Processing the Minimized or Fixed Variables

The previous example also shows that the order in which we processed the minimized and fixed variables matters.

In the example the search strategy of the algorithm is to first apply (C2) backwards, which fails for the sequent $r \overset{3}{\rightarrow} q, q \wedge \overset{3}{r} \rightarrow s, \neg q \overset{3}{\rightarrow} s \ \supset_{\overline{P};\overset{1}{\mathbf{q}},\overset{2}{r}} \overset{3}{s}$ (not shown in the proof). Then (C4) is applied backwards and we chose $\overset{1}{q}$ as principal formula. In the left premise we then try again to apply (C2) backwards, which fails again. Then again (C4) is applied backwards with $\overset{2}{r}$ as principal formula.

It is obvious that selecting $\overset{2}{r}$ as principal formula in the first backward application of (C4) would have been the better choice, since doing so would have lead to a successful backward application of (C2) in the left premise. However, although $\overset{1}{q}$ was a bad choice, the mechanism of use-check prevents us here from proving the right branch and thus somehow corrects our bad choice.

4.2.4 A Modified Calculus for Use-Check

According to the algorithm we now create a calculus PCC2 which bases on the calculus PCC and is intended for proof search. This calculus is necessary because the information we need for use-check can not be extracted from the standard sequents of CPC, CPRC and PCC.

A Modified Refutation Calculus

From the CPRC parts of our proofs, we need to know the anti-axiom which they use. We therefore extend the sequents used in the refutation calculus to hold the axiom of the proof. This is easily done by taking pairs of sequents instead of the sequent in the anti-axiom and by passing over this additional argument unchanged from the premise to the conclusion of every structural rule.

For the notation we use sequents holding this additional argument.

Definition 4.12 (CPRC2-sequent)
A CPRC2-sequent is a triple $\langle \mathrm{S}, \Gamma, \Delta \rangle$ denoted by $\mathrm{S}; \Gamma \supset \Delta$ where S is a CPC-sequent and Γ and Δ are multisets of formulas.

Definition 4.13 (refutable CPRC2-sequent)
A CPRC2-sequent $\mathrm{S}; \Gamma \supset \Delta$ is refutable if there exists a refutation of $\Gamma \supset \Delta$ having S as its anti-axiom.

$$\frac{}{\Gamma \supset \Delta; \Gamma \supset \Delta}(\mathsf{aax})^a$$

$$\frac{\mathrm{S}; \Gamma \supset \Delta, p}{\mathrm{S}; \Gamma, \neg \mathbf{p} \supset \Delta}(\neg \not\supset) \qquad \frac{\mathrm{S}; \Gamma, p \supset \Delta}{\mathrm{S}; \Gamma \supset \Delta, \neg \mathbf{p}}(\not\supset \neg)$$

$$\frac{\mathrm{S}; \Gamma, A, B \supset \Delta}{\mathrm{S}; \Gamma, \mathbf{A} \wedge \mathbf{B} \supset \Delta}(\wedge \not\supset) \qquad \frac{\mathrm{S}; \Gamma \supset \Delta, A, B}{\mathrm{S}; \Gamma \supset \Delta, \mathbf{A} \vee \mathbf{B}}(\not\supset \vee)$$

$$\frac{\mathrm{S}; \Gamma, A \supset \Delta}{\mathrm{S}; \Gamma, \mathbf{A} \vee \mathbf{B} \supset \Delta}(\cdot \vee \not\supset) \qquad \frac{\mathrm{S}; \Gamma \supset \Delta, A}{\mathrm{S}; \Gamma \supset \Delta, \mathbf{A} \wedge \mathbf{B}}(\not\supset \cdot \wedge)$$

$$\frac{\mathrm{S}; \Gamma, B \supset \Delta}{\mathrm{S}; \Gamma, \mathbf{A} \vee \mathbf{B} \supset \Delta}(\vee \cdot \not\supset) \qquad \frac{\mathrm{S}; \Gamma \supset \Delta, B}{\mathrm{S}; \Gamma \supset \Delta, \mathbf{A} \wedge \mathbf{B}}(\not\supset \wedge \cdot)$$

[a] $\Gamma \subseteq \mathcal{V} \cup \{\top\}, \Delta \subseteq \mathcal{V} \cup \{\bot\}, \Gamma \cap \Delta = \emptyset$

Figure 4.4: Deduction rules of CPRC2

The corresponding calculus (Figure 4.4) is equivalent to the calculus CPC because we just tag the sequents of a refutation with its axiom. Hence CPRC2 is sound and complete.

Definition 4.14 (count(Γ, A)**)**
Given a multiset of formulas Γ and a formula A we write count(Γ, A) to denote the number of occurrences of A in Γ.

Definition 4.15 (the calculus CPRC2)
We define the calculus CPRC2 to have the deduction rules given in Figure 4.4.

A Modified Circumscription Calculus

Regarding the circumscription rules we have to extend the sequents in such a way that the information we obtain from the CPC2 and CPRC2 deductions can be passed through the deduction rules. We thus extend the standard circumscription sequent with a use-set $\mathcal{D}$ and a set $\mathcal{S}$ of sequents representing the anti-axioms of the proof. Furthermore we need to use labeled formulas in order to apply use-check.

Definition 4.16 (PCC2-sequent)
A PCC2-sequent is a septuple $\langle \mathcal{S}, \mathcal{D}, \overline{\Sigma}, \overline{\Gamma}, \overline{\Delta}, \overline{P}, \overline{R} \rangle$ and is denoted with $\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$. It consists of two finite multisets $\overline{\Gamma} \subset \mathcal{L}_{\mathbb{N}}$ and $\overline{\Delta} \subset \mathcal{L}_{\mathbb{N}}$, three disjunctive sets of labeled variables $\overline{\Sigma}, \overline{P}$ and $\overline{R}$, a set of CPC-sequents $\mathcal{S}$ and a use-set $\mathcal{D} \subseteq$ labels$(\overline{\Gamma} \cup \overline{\Delta} \cup \overline{P} \cup \overline{R})$.

In order to make use of the additional information gained from the CPC2-

$$\frac{\mathsf{CPRC2} \vdash \mathrm{S}; \Gamma{\downarrow_{\mathbb{N}}}, \neg\overline{P}{\downarrow_{\mathbb{N}}} \supset p}{\{\mathrm{S}\}\,;\{\mathrm{n}\}\,;\overline{\Sigma}, \overset{\mathrm{n}}{p}; \overline{\Gamma} \supset_{\overline{P}} \overline{\Delta}}(\mathrm{C1}) \qquad \frac{\mathsf{CPC2} \vdash \mathcal{D}; \overline{\Sigma}, \overline{\Gamma} \supset \overline{\Delta}}{\emptyset; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}}(\mathrm{C2})$$

$$\frac{\mathcal{S}; \mathcal{D}; \overline{\Sigma}, \overset{\mathrm{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}}{\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P},\overset{\mathrm{n}}{p};\overline{R}} \overline{\Delta}}(\mathrm{C31})^{a} \qquad \frac{\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma}, \neg\overset{\mathrm{n}}{p} \supset_{\overline{P};\overline{R}} \overline{\Delta}}{\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P},\overset{\mathrm{n}}{p};\overline{R}} \overline{\Delta}}(\mathrm{C32})^{b}$$

$$\frac{\mathcal{S}_1; \mathcal{D}_1; \overline{\Sigma}, \overset{\mathrm{n}}{p}; \overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta} \quad \mathcal{S}_2; \mathcal{D}_2; \overline{\Sigma}; \overline{\Gamma}, \neg\overset{\mathrm{n}}{p} \supset_{\overline{P};\overline{R}} \overline{\Delta}}{\mathcal{S}_1 \cup \mathcal{S}_2; \mathcal{D}_1 \cup \mathcal{D}_2; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P},\overset{\mathrm{n}}{p};\overline{R}} \overline{\Delta}}(\mathrm{C3})^{c}$$

$$\frac{\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma}, \overset{\mathrm{n}}{r} \supset_{\overline{P};\overline{R}} \overline{\Delta}}{\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R},\overset{\mathrm{n}}{r}} \overline{\Delta}}(\mathrm{C41})^{d} \qquad \frac{\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma}, \neg\overset{\mathrm{n}}{r} \supset_{\overline{P};\overline{R}} \overline{\Delta}}{\mathcal{S}; \mathcal{D}; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R},\overset{\mathrm{n}}{r}} \overline{\Delta}}(\mathrm{C42})^{e}$$

$$\frac{\mathcal{S}_1; \mathcal{D}_1; \overline{\Sigma}; \overline{\Gamma}, \overset{\mathrm{n}}{r} \supset_{\overline{P};\overline{R}} \overline{\Delta} \quad \mathcal{S}_2; \mathcal{D}_2; \overline{\Sigma}; \overline{\Gamma}, \neg\overset{\mathrm{n}}{r} \supset_{\overline{P};\overline{R}} \overline{\Delta}}{\mathcal{S}_1 \cup \mathcal{S}_2; \mathcal{D}_1 \cup \mathcal{D}_2; \overline{\Sigma}; \overline{\Gamma} \supset_{\overline{P};\overline{R},\overset{\mathrm{n}}{r}} \overline{\Delta}}(\mathrm{C4})^{f}$$

[a] $\mathrm{n} \notin \mathcal{D}$ and $p \notin \Gamma$ for all $\Gamma \supset \Delta \in \mathcal{S}$.
[b] $\mathrm{n} \notin \mathcal{D}$.
[c] $\mathrm{n} \in \mathcal{D}_1$ or there exists $\Gamma \supset \Delta \in \mathcal{S}_1$ such that $p \in \Gamma$, and $\mathrm{n} \in \mathcal{D}_2$.
[d] $\mathrm{n} \notin \mathcal{D}$ and $\mathrm{count}(\Gamma, p) = 1$ for all $\Gamma \supset \Delta \in \mathcal{S}$.
[e] $\mathrm{n} \notin \mathcal{D}$ and $\mathrm{count}(\Delta, p) = 1$ for all $\Gamma \supset \Delta \in \mathcal{S}$.
[f] $\mathrm{n} \in \mathcal{D}_1$ or there exists $\Gamma \supset \Delta \in \mathcal{S}_1$ such that $\mathrm{count}(\Gamma, r) > 1$, and $\mathrm{n} \in \mathcal{D}_2$ or there exists $\Gamma \supset \Delta \in \mathcal{S}_2$ such that $\mathrm{count}(\Delta, r) > 1$.

Figure 4.5: Circumscription rules of PCC2

and CPRC2-sequents, we modify the circumscription rules to process this additional information.

Definition 4.17 (the calculus PCC2)
We define the calculus PCC2 to have the deduction rules given in Figure 4.5.

Regarding the non-branching circumscription rules the modifications are more or less straightforward.

(C1) We initialize the set of anti-axioms with the anti-axiom of the refutation of the premise and register the label of the principal constraint in the use-set of the conclusion.

(C2) We take over the use-set $\mathcal{D}$ from the CPC2-sequent and set an empty set of anti-axioms.

Regarding the branching circumscription rules we need — as in the calculus CPC2 — three rules that replace a branching rule. One for the case where

both premises need to be proved and two rules for the cases where one of the premises can be omitted. The conditions to apply the rules are given in the footnotes of Figure 4.5. Since we use the use-set $\mathcal{D}$ to test whether a variable was used in a classical axiom and whether a minimized variable was used as principal constraint in a (C1)-rule application, the conditions are a little bit easier than in the corresponding theorems.

(C31) This rule reflects Theorem 4.8. The conditions to apply the rule are accordingly. That is n must not be in $\mathcal{D}$ (conditions 1 and 3) and there is no occurrence of p in the antecedent of any element of $\mathcal{S}$ (condition 2).

(C32) This rule reflects Theorem 4.9. The conditions to apply the rule are accordingly. That is n must not be in $\mathcal{D}$.

(C3) This rule reflects the case where use-check can not be applied. As conditions to apply the rule we thus have that n is in $\mathcal{D}_1$ or p occurs in the antecedent of an element of $\mathcal{S}_1$ (negated conditions of (C31)) and that n is in $\mathcal{D}_2$ (negated condition of (C32)).

(C41) This rule reflects Theorem 4.10. The conditions to apply the rule are accordingly. That is n must not be in $\mathcal{D}$ (condition 1) and there is no multiple occurrence of p in the antecedent of any element of $\mathcal{S}$ (condition 2).

(C42) This rule reflects Theorem 4.11. The conditions to apply the rule are accordingly. That is n must not be in $\mathcal{D}$ (condition 1) and there is no multiple occurrence of p in the succedent of any element of $\mathcal{S}$ (condition 2).

(C4) This rule reflects the case where use-check can not be applied. As conditions to apply the rule we thus have that n is in $\mathcal{D}_1$ or p occurs manifold in the antecedent of an element of $\mathcal{S}_1$ (negated conditions of (C41)) and that n is in $\mathcal{D}_2$ or p occurs manifold in the succedent of an element of $\mathcal{S}_2$ (negated conditions of (C42)).

Soundness and Completeness of PCC2

The calculus PCC2 is sound and complete. Soundness is obtained from the theorems 4.8–4.11. Since we can convert any proof of PCC into a proof of PCC2 by labeling the formulas in the circumscription and classical sequents with an arbitrary value n, completeness is also easily obtained.

Theorem 4.18 (soundness of PCC2)
If $\mathcal{S};\mathcal{D};\overline{\Sigma};\overline{\Gamma} \supset_{\overline{P};\overline{R}} \overline{\Delta}$ is deducible in PCC2 then $\overline{\Sigma}{\downarrow_{\mathbb{N}}};\overline{\Gamma}{\downarrow_{\mathbb{N}}} \supset_{\overline{P}\downarrow_{\mathbb{N}};\overline{R}\downarrow_{\mathbb{N}}} \overline{\Delta}{\downarrow_{\mathbb{N}}}$ is valid.

Proof. Follows directly from the theorems 4.8–4.11. □

Theorem 4.19 (completeness of PCC2)
If $\Sigma; \Gamma \supset_{P;R} \Delta$ is valid then there exists a set of anti-axioms $\mathcal{S}$ such that $\mathcal{S}; \{n\}; \overset{\text{n}}{\Sigma}; \overset{\text{n}}{\Gamma} \supset_{\overset{\text{n}}{P};\overset{\text{n}}{R}} \overset{\text{n}}{\Delta}$ is deducible in PCC2 for any label n.

Proof. Let $\Sigma; \Gamma \supset_{P;R} \Delta$ be valid and $\mathcal{P}$ be a PCC-proof of it. We show our claim by induction on the number c of circumscription rule applications in $\mathcal{P}$.

- $c = 1$:

  Then the last rule of $\mathcal{P}$ was either (C1) or (C2):

  – $\dfrac{\mathsf{CPRC} \vdash \Gamma, \neg P \supset p}{\Sigma', p; \Gamma \supset_P \Delta}$(C1)

  If S is the anti-axiom of the refutation of $\Gamma, \neg P \supset p$ then the CPRC2-sequent $\mathrm{S}; \Gamma, \neg P \supset p$ is refutable in CPRC2. Using (C1) we can deduce $\{n\}; \mathrm{S}; \overset{\text{n}}{\Sigma'}, \overset{\text{n}}{p}; \overset{\text{n}}{\Gamma} \supset_{\overset{\text{n}}{P}} \overset{\text{n}}{\Delta}$ for any label n.

  – $\dfrac{\mathsf{CPC} \vdash \Sigma, \Gamma \supset \Delta}{\Sigma; \Gamma \supset_{P;R} \Delta}$(C2)

  Then $\{n\}; \overset{\text{n}}{\Sigma}, \overset{\text{n}}{\Gamma}, \supset \overset{\text{n}}{\Delta}$ is provable in CPC2 for any label n. Using (C2) we obtain a proof of $\{n\}; \emptyset; \overset{\text{n}}{\Sigma}, \overset{\text{n}}{p}; \overset{\text{n}}{\Gamma} \supset_{\overset{\text{n}}{P};\overset{\text{n}}{R}} \overset{\text{n}}{\Delta}$.

- $c > 1$:

  Then the last rule of $\mathcal{P}$ was either (C3) or (C4). Since both cases are similar to prove we only show one case:

  – $\dfrac{\Sigma, p; \Gamma \supset_{P';R} \Delta \quad \Sigma; \Gamma, \neg p \supset_{P';R} \Delta}{\Sigma; \Gamma \supset_{P',p;R} \Delta}$(C3) with $P = P' \cup \{p\}$.

  By induction hypothesis there exist two set of anti-axioms $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\{n\}; \mathcal{S}_1; \overset{\text{n}}{\Sigma}, \overset{\text{n}}{p}; \overset{\text{n}}{\Gamma} \supset_{\overset{\text{n}}{P'};\overset{\text{n}}{R}} \overset{\text{n}}{\Delta}$ and $\{n\}; \mathcal{S}_2; \overset{\text{n}}{\Sigma}, \overset{\text{n}}{\neg p}; \overset{\text{n}}{\Gamma} \supset_{\overset{\text{n}}{P'};\overset{\text{n}}{R}} \overset{\text{n}}{\Delta}$ are provable in PCC2 for any label n. Using (C3) we obtain a proof of $\{n\}; \mathcal{S}_1 \cup \mathcal{S}_2; \overset{\text{n}}{\Sigma}, \overset{\text{n}}{p}; \overset{\text{n}}{\Gamma} \supset_{\overset{\text{n}}{P};\overset{\text{n}}{R}} \overset{\text{n}}{\Delta}$.

□

4.3 Experimental Results

In this section we compare the different proving approaches with different stages of enhancement on some scalable circumscription problems. The problems mainly aim to point out the advantages and disadvantages of the different approaches and enhancements. All problems were computed under Debian Linux 4.0 on an AMD Sempron 2600+ with 1.5 GB RAM.

To prove whether a formula A is $(P;R)$-minimally entailed by Γ we have investigated three main approaches.

1. Use the CPC prover to prove the sequent $\mathrm{CIRC}(\Gamma, P, Q) \supset A$, where $Q := \mathrm{vars}(\Gamma) \setminus (P \cup R)$ (cf. definition 3.12 on page 60). For calculating the circumscription theory, we apply enhancement 1 from section 4.1. When referring to this approach we speak of the prover PRc.
   As variants we have also looked at the other enhancements given in Section 4.1. But for the sequent for which we have investigated this approach, those enhancements were of little use or even lead to worse results.

2. Do backward application of the deduction rules and prefer to apply branching rules (cf. Algorithm 3 on page 70). For this approach we have investigated the prover PR1, implementing no improvements, and PR1u, implementing use-check.

3. Do backward application of the deduction rules and prefer to apply non-branching rules (cf. Algorithm 4 on page 71).
   For this approach we have taken the general improvements and use-check into account. To show the impact of the different improvements we have investigated the provers PR2, implementing no improvements, PR2g, implementing the general improvements, and PR2gu, implementing the general improvements and use-check.

We also compare our implementation with the prover mm, a prolog implementation of Niemelä's tableau calculus for minimal models [22]. Now mm is restricted to theories consisting of clauses, therefore some problems that we investigate can not be specified in the same short form for mm. A direct comparison with mm is thus not possible for those problems.

4.3.1 Problem 1: A Single Minimal Model

We start with a very simple theory that is defined as follows.

$$
\begin{aligned}
T_1(n) &:= \{q_i, p_i \vee q_i : 1 \leq i \leq n\} \\
P_1(n) &:= \{p_i : 1 \leq i \leq n\} \\
Q_1(n) &:= \{q_i : 1 \leq i \leq n\} \\
R_1(n) &:= \emptyset
\end{aligned}
$$

An interpretation I is then a model of $T_1(n)$ if and only if $Q_1(n) \subseteq I$. The interpretation of the variables in $P_1(n)$ is arbitrary for any model of $T_1(n)$. If we minimizing $P_1(n)$ then $M := Q_1(n)$ is the only $P_1(n)$-minimal model of $T_1(n)$.

Proving $T_1(n) \supset_{P_1(n)} \bigvee \neg P_1(n)$

The first problem we investigate is that the following formula is $P_1(n)$-minimally entailed by $T_1(n)$.

$$A_1(n) := \bigvee \neg P_1(n)$$

This should be rather easy to accomplish since we only have to show that the negation of an arbitrary minimized variable is valid in the single minimal model.

When using PRc to prove $T_1(n) \supset_{P_1(n)} A_1(n)$ we examine that the proving time grows exponentially with n (cf. Figure 4.6). Proving $T_1(n+1) \supset_{P_1(n+1)} A_1(n+1)$ takes about four times as long as proving $T_1(n) \supset_{P_1(n)} A_1(n)$. This is quite what could be expected since with each variable that we minimize or vary, we double the size of the circumscription theory. And by increasing the problem size by one we add one minimized and one varied variable, hence the factor of 4.

With the prover PR1 we have slightly better results, but we still have strong exponential growth. This is also what could be expected since the number of nodes of the search tree in which we start to apply (C1) and (C2) backwards is doubled when incrementing the problem size by 1.

The prover PR1u performs better than PRc and PR1 but is far from the performance of the other provers (see below). The reason for this is that in the classical axioms the prover chooses different principal formulas and hence use-check does not work in an optimal way. Still there are a lot of nodes where use-check can be applied. For $n = 50$ for example use-check is successful in 791.421 of 908.346 backward applications of (C4). But this is not enough to come close to the performance of the other provers.

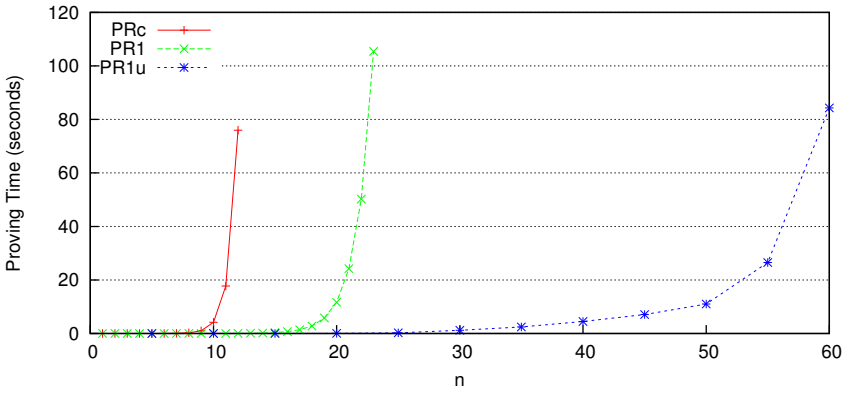


Figure 4.6: Proving time of $T_1(n) \supset_{P_1(n)} A_1(n)$

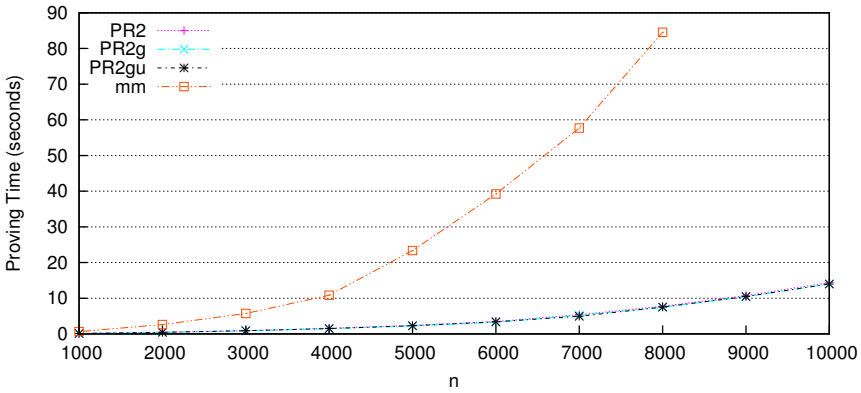


Figure 4.7: Proving time of $T_1(n) \supset_{P_1(n)} A_1(n)$

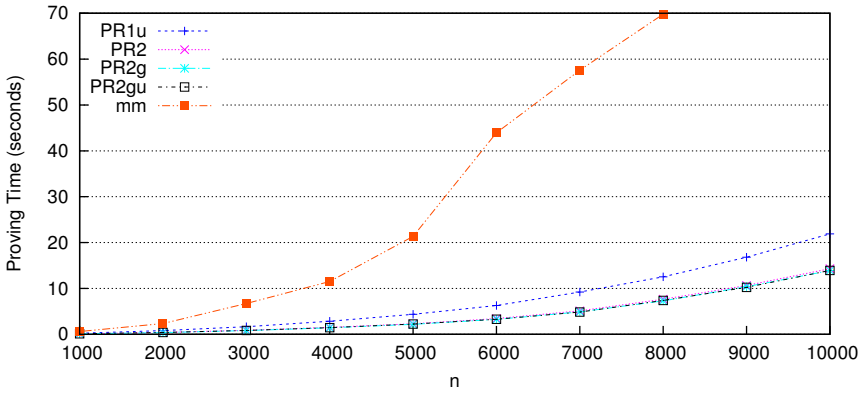


Figure 4.8: Proving time of $T_1(n) \supset_{P_1(n)} \neg p_1$

As already indicated we have a much better behavior if we use a prover that prefers non-branching to branching rules (cf. Figure 4.7).

This is not really astonishing if we look at the proof that we find with our algorithms. It is, independently of the problem size, of the following form.

$$\dfrac{\dfrac{\vdots}{\mathsf{CPRC} \vdash T_1(n), \neg p_2, \ldots, \neg p_n \supset p_1}(\mathrm{C1}) \quad \dfrac{\vdots}{\mathsf{CPC} \vdash T_1(n), \neg p_1 \supset A_1(n)}(\mathrm{C2})}{\dfrac{p_1; T_1(n) \supset_{p_2,\ldots,p_n} A_1(n)}{T_1(n) \supset_{p_1,\ldots,p_n} A_1(n)}(\mathrm{C3})}$$

A closer look at the algorithms shows that for this example there are very few backward applied circumscription rules that fail. For PR2 we have two backward applications of (C2) that fail until we find a proof. For the other provers we have, due to the easy general improvements, only one failed backward application of (C2). Since all provers find the same proof it is not astonishing that they all show more or less the same performance.

The prover mm also shows a good performance, but it is clearly worse than the provers that prefer non-branching to branching rules.

Proving $T_1(n) \supset_{P_1(n)} \neg p_k$

Instead of proving that the negation of an arbitrary minimized variable is valid in our minimal model we can also prove the validity of the negation of a specific minimized variable p_k.

For $k = 1$ the provers PR2, PR2u and PR2gu show the same performance as for the previous problem (cf. Figure 4.8). The prover PR1u also shows good results and comes close to the performance of the other provers because of

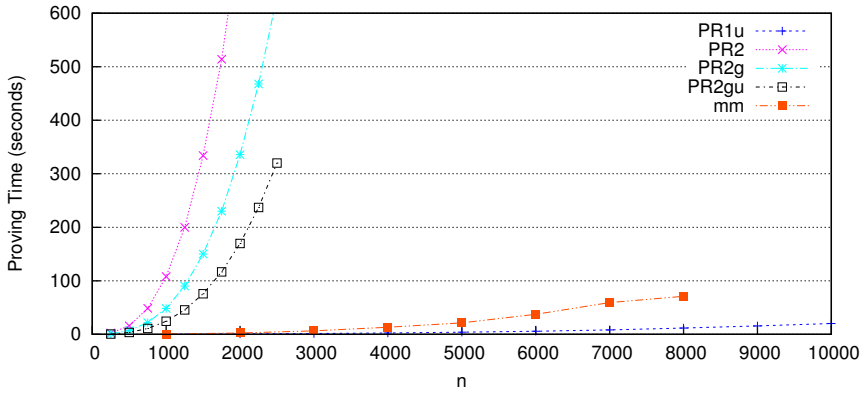


Figure 4.9: Proving time of $T_1(n) \supset_{P_1(n)} \neg p_n$

optimal use-check (for $n = 10.000$ use-check is successful in 19.998 of 19.999 branchings). Though the prover mm does not reach the proving times of our provers it shows a good performance.

For $k = n$ the picture changes completely. While PR1u still performs as good as for $k = 1$ the other provers perform definitely worse (cf. Figure 4.9). The prover mm is also stable and thus clearly better than PR2, PR2g and PR2gu but not as good as PR1u.

The reason for the loss of performance is up to the order in which the minimized variables are processed. Our provers start by applying (C3) with principal formula p_1 backwards. Then they try to backward apply (C2) before continuing with (C3) and principal formula p_2. We thus have for all provers considerably more failed (C2) backward applications and this results in a clear loss of performance.

Compared to the results of the previous problem we now also have differences in the performance of PR2, PR2u and PR2gu. PR2 is clearly slower than PR2g which itself performs worse than PR2gu. Looking at the prover statistics we see that for $n = 1.000$ the general improvements allow us to omit 999 backward applications of (C1) and one of (C2). With use-check we can omit 999 times proving the second premise of (C3) which results that in comparison to PR2g only 1.001 instead of 2.000 backward applications of (C2) are processed. Therefore PR2gu is almost twice as fast as PR2g.

4.3.2 Problem 2: Fixed Variables

For the theory of the previous section we obtain the same single minimal model if we set the non minimized variables to be fixed. However, with fixed

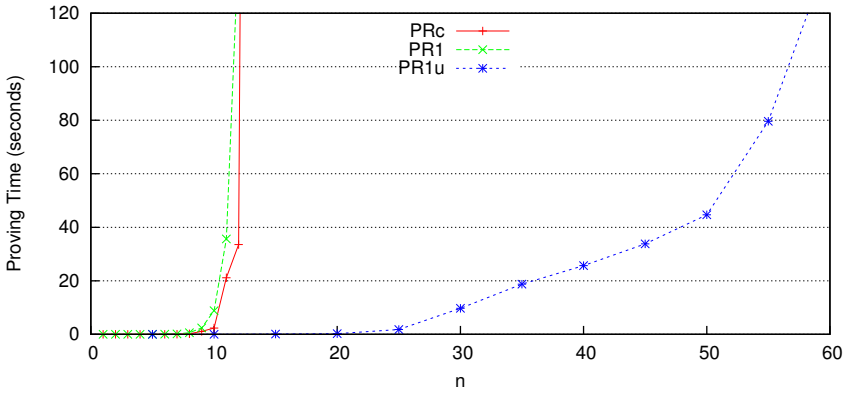


Figure 4.10: Proving time of $T_2(n) \supset_{P_2(n);R_2(n)} A_2(n)$

variables we have a lot more of work to do for backward proof search. We investigate this with the following theory.

$$\begin{aligned} T_2(n) &:= \{\neg\neg r_i, p_i \vee r_i : 1 \leq i \leq n\} \\ P_2(n) &:= \{p_i : 1 \leq i \leq n\} \\ Q_2(n) &:= \emptyset \\ R_2(n) &:= \{r_i : 1 \leq i \leq n\} \end{aligned}$$

Besides the change in the fixed variables there is another minor modification in our theory, it contains $\neg\neg r_i$ instead of r_i. The reason for this is that we want to avoid that the following simple improvement in our algorithms succeeds.

> When applying (C4) backwards, we check whether the formula (a possibly negated fixed variable) that we want to add to our sequent, is already present. If it is, we simply ignore that fixed variable and proceed with the next fixed or minimized variable.

The formula we prove is the same as in the first example.

$$A_2(n) := \bigvee \neg P_2(n)$$

For the prover PRc it could be expected that the performance is now better since there are no more varied variables and hence $\mathrm{CIRC}(T_2(n), P_2(n), \emptyset)$ should be easier to calculate. Nonetheless we have about the same proving times as with varied variables (cf. Figure 4.10). A closer look at the behavior of the prover reveals that our expectation is not wrong. Computing $\Gamma_1 := \mathrm{CIRC}(T_1(11), P_1(11), Q_1(11))$ takes about 18 seconds while computing $\Gamma_2 :=$

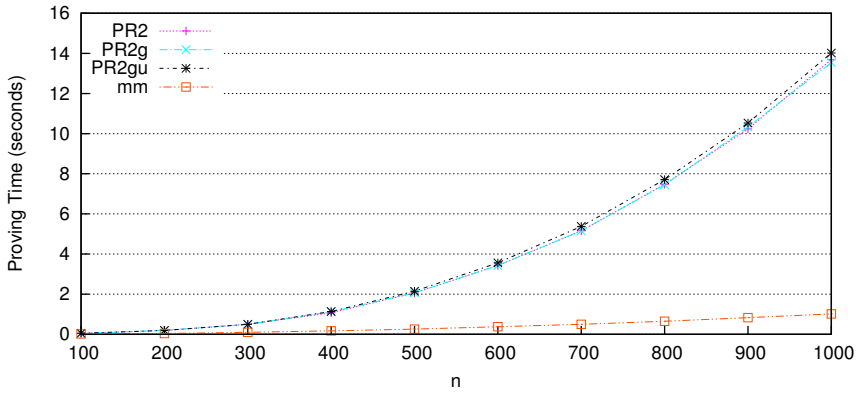


Figure 4.11: Proving time of $T_2(n) \supset_{P_2(n);R_2(n)} A_2(n)$

$\mathrm{CIRC}(T_2(11), P_2(11), \emptyset))$ takes only 5 seconds. It is the complexity of the resulting circumscription theory that matters. For our example we have $\sum\{\mathrm{len}(A) : A \in \Gamma_1\} = 56.331$ while $\sum\{\mathrm{len}(A) : A \in \Gamma_2\} = 2.805.033$. The consequence of this is that proving $\Gamma_1 \supset A_1(n)$ takes only a split second while proving $\Gamma_2 \supset A_2(n)$ takes about 16 seconds.

The prover PR1 shows a worse performance. While for no fixed variables PR1 could prove the test sequent in less than a minute for $n = 22$, it now already breaks the one minute limit for $n > 11$. This is not astonishing. By increasing the problem size from n to $n+1$ we now not only add an additional minimized but also an additional fixed variable. The consequence of that is that we no longer double but quadruple the number of nodes in which we start to backward apply (C1) and (C2).

Compared to PRc and PR1, the prover PR1u now shows the best performance. The reason is that it can successfully apply use-check in backwards application of (C4). For $n = 10$ for example we can omit proving the second branch in 45 of 55 (C4) nodes and backward apply (C3) 11.263 times. Compared to PR1 which has 1.023 (C4) and 1.047.552 (C3) backwards applications this is quite an improvement.

The proving times of PR2, PR2g, PR2gu and mm are given in Figure 4.11. While mm performs best[3], the other provers show about the same performance. They are clearly faster than PR1u but significantly slower than for the theory without fixed variables. Obviously use-check is with the chosen proving strategy not able to detect that having fixed variables is superfluous. The reason for this can be seen on the proof for $n = 1$.

[3]For mm specifying a double negation on the fixed variables was not possible, we thus operated on the theory $T_2(n) := \{r_i, p_i \vee r_i : 1 \leq i \leq n\}$.

$$\dfrac{\dfrac{\dfrac{\mathsf{CPC} \vdash \neg p, r, r, r \vee p \supset \neg p}{\neg p, r, r, r \vee p \supset \neg p}(\text{C2}) \quad \dfrac{\mathsf{CPRC} \vdash r, r, r \vee p \supset p}{p; r, r, r \vee p \supset \neg p}(\text{C1})}{r, r, r \vee p \supset_p \neg p}(\text{C3}) \quad \dfrac{\mathsf{CPC} \vdash \neg r, r, r \vee p \supset \neg p}{\neg r, r, r \vee p \supset_p \neg p}(\text{C2})}{r, r \vee p \supset_{p;r} \neg p}(\text{C4})$$

In the left most branch we end up in a (C3) rule branching to a (C2) and (C1) rule. In the corresponding anti-axiom, all fixed variables occur twice, therefore use-check can not succeed for any (C4) rule. Processing the right premise of a (C4) rule immediately leads to a successful backward application of (C2). The number of necessary non-branching rules thus linearly grows with the problem size. Our analysis is confirmed by the statistics of the prover. For $n = 1.000$ we have 1 successfully processed (C1) rule and 2.002 processed (C2) rules of which 1.001 fail. There are no second premises that can be omitted through use-check.

The question arises if it is not possible to filter out those fixed variables which are known to have the same value in all models of a theory. In order to do so we would need a preprocessing step which checks for each fixed variable whether the variable itself or its negation is a classically consequence of the antecedent of the circumscription sequent. However, the time used for such a preprocessing step would be more or less equivalent to the additional time we need when not doing the preprocessing. Hence there was no reason for us to implement this.

4.3.3 Problem 3: Fixed Variables Revisited

In this section we present an example where use-check is successful in (C4) applications for provers that prefer non-branching to branching rules. In the previous example the fixed variables were part of the theory, had thus a multiple occurrence in the anti-axioms and therefore use-check never succeeded. In the following theory this is not case.

$$\begin{aligned} T_3(n) &:= \{r_{2i-1} \vee r_{2i}, p_i \vee r_{2i-1} \vee r_{2i} : 1 \leq i \leq n\} \\ P_3(n) &:= \{p_i : 1 \leq i \leq n\} \\ Q_3(n) &:= \emptyset \\ R_3(n) &:= \{r_i : 1 \leq i \leq 2n\} \end{aligned}$$

A model M of $T_3(n)$ is $(P_3(n); R_3(n))$-minimal, if $P \cap M = \emptyset$ and $r_{2i-1} \in M$ or $r_{2i} \in M$ for $1 \leq i \leq n$. $T_3(n)$ has thus 3^n $(P_3(n); R_3(n))$-minimal models.

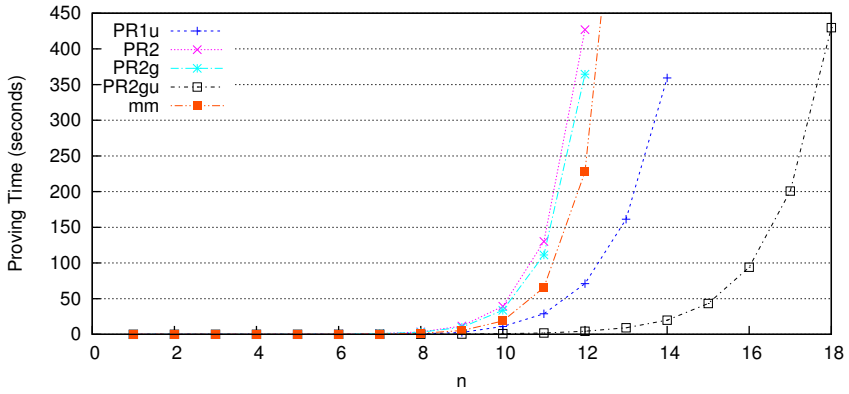


Figure 4.12: Proving time of $T_3(n) \supset_{P_3(n);R_3(n)} A_3(n)$

The formula we prove is the same as in the first example.

$$A_3(n) := \bigvee \neg P_3(n)$$

The results are given in figure 4.12. First of all we see that the proving time already raises considerably with a small problem size. A cause for this is that most of the right premises of the (C4) rules are no longer easily provable with a (C2) applications. Hence the proof tree branches out much heavier.

The provers without use-check behave worst. Although the general improvements are of some use they don't make a great difference. For $n = 12$ they are good to omit 531.441 of 2.657.203 backward applications of (C2), none of the 531.441 (C1) applications can be omitted.

PR1u performs better. Compared to the 797.160 (C4) rules that PR2 and PR2g need, it only processes 20.451 such rules and can omit proving the second premise in 12.261 cases. However, it needs much more (C3) rules. PR2 and PR2g process 531.441 of them while PR1u has to fully process 1.089.454 out of 6.389.078.

PR2gu performs best. It needs 12.285 (C4) rules and can omit proving the second branch in 4.095 cases. Of the 4.096 processed (C3) rules it can never omit proving the second premise.

The prover mm performs better than the prover with no or only general improvements, is not as good as PR1u and clearly worse than PR2u.

4.3.4 Problem 4: Number of Minimal Models Grow Linearly

With the next theory we are going to investigate use-check in (C3) for a theory whose number of minimal models linearly grows with the problem

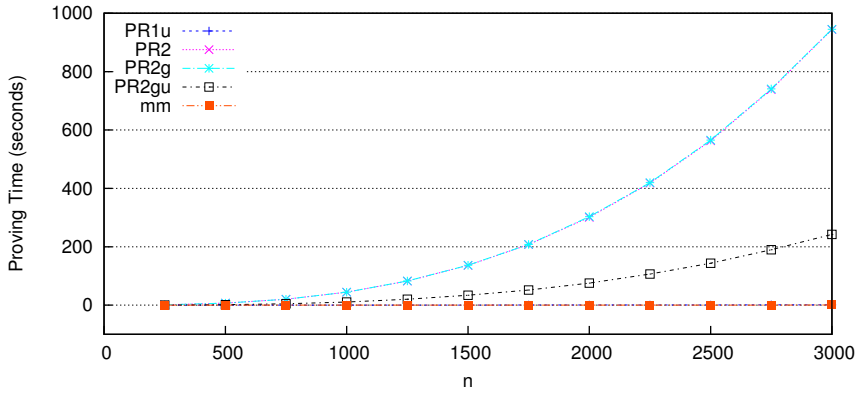


Figure 4.13: Proving time of $T_4(n) \supset_{P_4(n)} A_4(n)$

size.

$$
\begin{aligned}
T_4(n) &:= \bigvee \{p_i : 1 \leq i \leq n\} \\
P_4(n) &:= \{p_i : 1 \leq i \leq n\} \\
Q_4(n) &:= \emptyset \\
R_4(n) &:= \emptyset
\end{aligned}
$$

The models of $T_4(n)$ are exactly those that contain at least one variable of $P_4(n)$. The $P_4(n)$-minimal models of $T_4(n)$ are thus those that contain exactly one $p \in P_4(n)$, i.e. $M_1 = \{p_1\}, \ldots, M_n = \{p_n\}$.

Proving $T_4(n) \supset_{P_4(n)} \neg(p_1 \wedge p_n)$

The first problem we investigate is that no minimal model contains p_1 and p_n

$$A_4(n) := \neg(p_1 \wedge p_n)$$

The results (cf. Figure 4.13) resemble those of proving $T_1(n) \supset_{P_1(n)} p_n$ and in spite of the growing number of minimal models we have in fact more or less the same situation. mm performs best, PR1u is close behind. They both need less than a second to prove the problem for $n = 3000$.

Let us look at the prover statistics for $n = 1.000$. With the general improvements we can halve the number of (C1) rules. PR2 processes 1.998 such rules while PR2g gets along with 1.000. For PR1u and PR2gu use-check works in an optimal way for (C3). PR1u can omit proving the second branch in 1.996

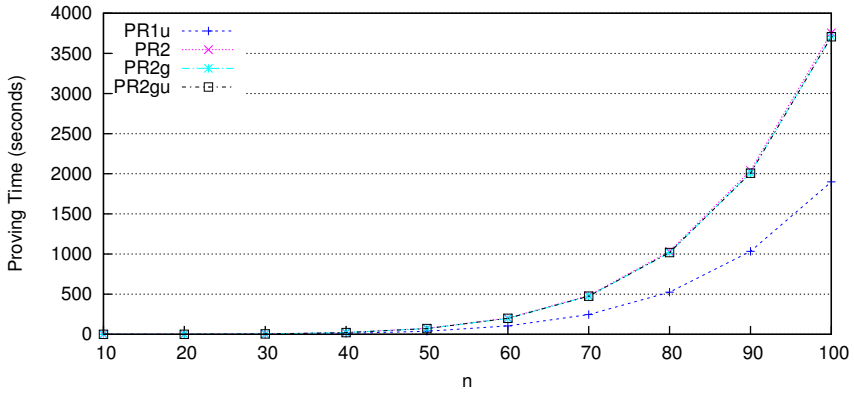


Figure 4.14: Proving time of $T_4(n) \supset_{P_4(n)} B_4(n)$

of 1.999 cases, PR2gu can do so in 999 of 1.000 branching rules. The reason why PR2gu performs worse than PR1u is again to find in its strategy to prefer non-branching rules. For PR2gu this leads to 999 unsuccessful backward applications of (C2) while in PR1u only one is not successful.

Proving $T_4(n) \supset_{P_4(n)} \bigwedge \{\neg(p_i \wedge p_j) : 1 \leq i < j \leq n\}$

The next problem we investigate is that at most one minimized variable is valid in every minimal model.

$$B_4(n) := \bigwedge \{\neg(p_i \wedge p_j) : 1 \leq i < j \leq n\}$$

The results are given in Figure 4.14. All provers preferring non-branching rules show about the same performance. The reason for this is that with the chosen proving strategy we can only omit processing about n (C1) and n (C2) rules. In addition use-check can never successfully be applied. Nevertheless, for $n = 50$ only 1.275 of the $2^{50} - 1$ potential branchings are processed. This is mainly due to the proving strategy. If the set of constraints contains two elements, backward application of (C1) is successful. We can thus cut off a big part of the search tree such that the number of processed branching rules only grows quadratically to the problem size.

PR1u performs best and needs about half as long as the other provers. One of the reasons is that there are only half as many backward applied (C2) rules than in the other provers (for $n = 50$ it is 1.225 compared to 2.499). The other reason is that use-check is again very successful. For $n = 50$ the prover encounters 20.875 branchings but can omit in 19.600 cases proving the second premise. Compared to the other provers there is thus just one additional branching rule which it has to fully process.

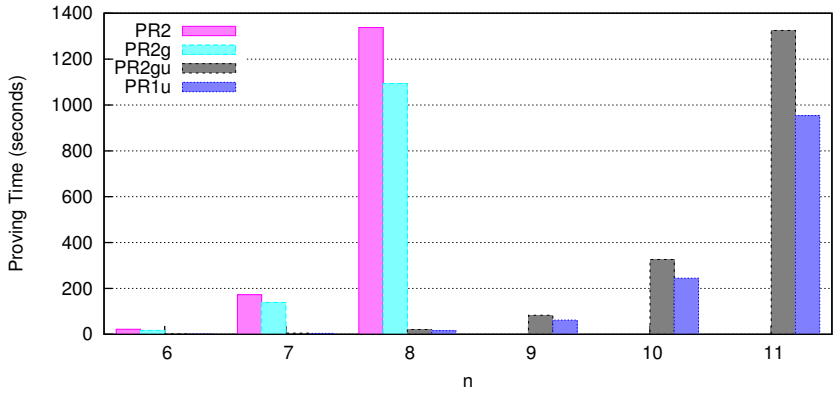


Figure 4.15: Proving time of $T_5(n) \supset_{P_5(n)} A_5(n)$

Since $B_4(n)$ is not in disjunctive normal form, it can not be specified this way in mm. A direct comparison is therefore not possible.

4.3.5 Problem 5: Number of Minimal Models Grow Exponentially

For the next theory we are testing, the number of minimal models exponentially grows with the problem size. The theory is as follows.

$$\begin{aligned} T_5(n) &:= \{p_{3i-2} \vee p_{3i-1} \vee p_{3i} : 1 \leq i \leq n\} \\ P_5(n) &:= \{p_i : 1 \leq i \leq 3n\} \\ Q_5(n) &:= \emptyset \\ R_5(n) &:= \emptyset \end{aligned}$$

For $p_{3i-2} \vee p_{3i-1} \vee p_{3i}$ to be valid, either p_{3i-2}, p_{3i-1} or p_{3i} must be valid. Since $T_5(n)$ contains n formulas of this form, there exist 8^n models of $T_5(n)$. The $(P_5(n))$-minimal models are then those, that contain exactly one element of $\{p_{3i-2}, p_{3i-1}, p_{3i}\}$ for $1 \leq i \leq n$. $T_5(n)$ thus has 3^n minimal models.

The first problem that we are going to investigate is that no minimal model contains p_{3i-2} and p_{3i-1}.

$$A_5(n) := \bigwedge \{\neg(p_{3i-2} \wedge p_{3i-1}) : 1 \leq i \leq n\}$$

The results are given in Figure 4.15. We see that proving time increases very rapidly such that the reasonable range of the problem size that can

be examined is quite small. This is clearly an effect of the exponentially growing number of models.

PR2g is about 20 percents faster than PR2 due to the general improvements. For $n = 7$ they allow us to reduce the number of failed (C1) applications from 459.679 to 137.983 and the number of failed (C2) applications from 781.379 to 643.393. Both provers process about 22 percent of the potentially branchings rules. For $n = 7$ that are 459.682 of 2.097.151.

The provers with use-check show much better performance. For $n = 7$ the prover PR2gu gets along with 11.662 branchings rules and can omit 2.733 times proving the second premise.

PR1u encounters 25.716 such rules and can omit proving the second branch in 13.309 cases. It thus has to fully process 3.478 more branching rules than PR2gu. Nevertheless it is faster and this is again to the intermediate non-branching rules which it does not apply. It therefore processes 5.103 (C1) and 12'408 (C2) rules while PR2gu encounters 8927 (C1) and 16'785 (C2) rules.

As for the previous problem we can not make a direct comparison with mm since $B_5(n)$ is not in disjunctive normal form.

Changing Order of Processed Variables

How do the provers behave if we change in this example the order in which he minimized variables are processed. We can investigate this easily with a minor change of the formula we prove.

$$B_5(n) := \bigwedge \{\neg(p_{3i-1} \wedge p_{3i}) : 1 \leq i \leq n\}$$

The results are given in Figure 4.16. The provers without use-check are marginally slower in proving $B_5(n)$ because they now process a small percentage of additional circumscription rules.

Both provers with use-check show quite a drop of performance. The minor change in the proven formula causes use-check to be considerable less effective. For proving $B_5(n)$ both provers now process about 11 times as many (C1), (C2) and (C3) rules as for proving $A_5(n)$.

4.3.6 Problem 6: Graph Problem

Dix, Furbach and Niemelä [9] suggest to produce benchmarks from graph problems. The problems they suggest are given in terms of normal logic

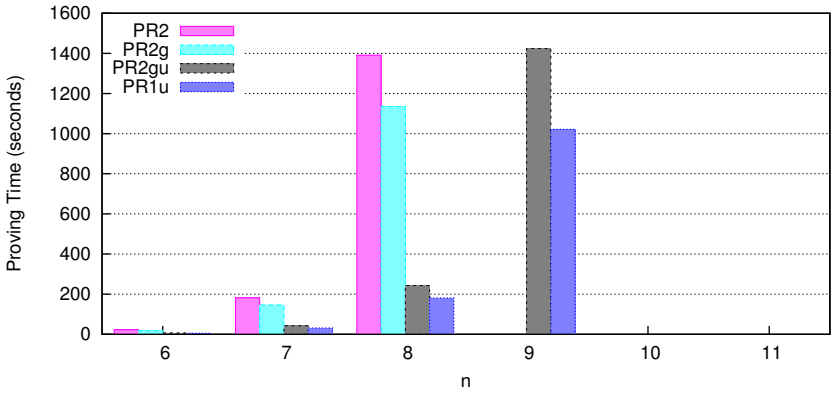


Figure 4.16: Proving time of $T_5(n) \supset_{P_5(n)} B_5(n)$

programs with the stable model semantics. We investigate the Hamiltonian circuit problem[4] specified by them as follows.

> *As an example where the expressivity of nonmonotonic logics provides compact representation we consider the Hamiltonian circuit problem, i.e., the problem of finding a path that visits each vertex of a given graph exactly once and returns to the starting vertex. The corresponding logic program is constructed as follows: (i) a set of atomic fact of the form* $vertex(v)$ *and* $arc(v, u,)$ *corresponding to the vertices and arcs is taken, (ii) the following rules are included*
>
> $$hcircuit(V_1, V_2) \leftarrow arc(V_1, V_2), not\, otherroute(V_1, V_2)$$
> $$otherroute(V_1, V_2) \leftarrow arc(V_1, V_2), hcircuit(V_1, V_3), not(V_2 = V_3)$$
> $$otherroute(V_1, V_2) \leftarrow arc(V_1, V_2), hcircuit(V_3, V_2), not(V_1 = V_3)$$
> $$reached(V_2) \leftarrow hcircuit(V_1, V_2), reached(V_1)$$
> $$reached(V_2) \leftarrow hcircuit(V_1, V_2), initialnode(V_1)$$
> $$noncircuit \leftarrow vertex(V), not\, reached(V)$$
> $$p \leftarrow not\, p, noncircuit$$
>
> *and (iii) one of the vertices* v *is taken as the starting vertex (*$initialnode(v)$ *is added). The resulting program has a stable model if and only if the graph has a Hamiltonian circuit. Again, a stable model of the program provides directly a Hamiltonian circuit by the facts of the form* $hcircuit(v, u)$ *true in the model A benchmark for query-evaluation can be obtained by removing the last rule. Then the graph has a Hamiltonian circuit if and*

[4] A Hamiltonian circuit is a route through a directed graph $\mathcal{G}$ that visits every vertex of $\mathcal{G}$ and ends at its starting point.

only if the resulting program has a stable model not containing noncircuit.

Their logic program can be directly translated into propositional logic:

Let $\mathcal{G} := (\mathsf{V}, \mathsf{E})$ be a finite directed graph with $\mathsf{V} \in \mathbb{N}$ and $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$. Our language then consists of the following variables.

- $\{\mathsf{hcircuit}_{ij} : (i,j) \in \mathsf{E}\}$: The edge (i,j) is part of our route.
- $\{\mathsf{otherroute}_{ij} : (i,j) \in \mathsf{E}\}$: The edge (i,j) is not part of our route.
- $\{\mathsf{reached}_i : i \in \mathsf{V}\}$: Vertex i is visited by our route.
- $\{\mathsf{initialnode}_i : i \in \mathsf{V}\}$: Our route starts at vertex i.
- noncircuit: Our route is not a hamilton circuit.

The problem of a hamilton circuit is then define as follows.

$$
\begin{aligned}
T_{\mathsf{hc}}(\mathcal{G}) := &\{\neg\mathsf{otherroute}_{ij} \to \mathsf{hcircuit}_{ij} : (i,j) \in \mathsf{E}\} && (4.1)\\
&\cup \{\mathsf{hcircuit}_{ij} \to \mathsf{otherroute}_{ik} : (i,j),(i,k) \in \mathsf{E} \text{ and } j \neq k\} && (4.2)\\
&\cup \{\mathsf{hcircuit}_{ij} \to \mathsf{otherroute}_{kj} : (i,j),(k,j) \in \mathsf{E} \text{ and } i \neq k\} && (4.3)\\
&\cup \{\mathsf{hcircuit}_{ij} \wedge \mathsf{reached}_i \to \mathsf{reached}_j : (i,j) \in \mathsf{E}\} && (4.4)\\
&\cup \{\mathsf{hcircuit}_{ij} \wedge \mathsf{initialnode}_i \to \mathsf{reached}_j : (i,j) \in \mathsf{E}\} && (4.5)\\
&\cup \{\neg\mathsf{reached}_i \to \mathsf{noncircuit} : i \in \mathsf{V}\} && (4.6)
\end{aligned}
$$

The intended meaning of the formulas is straight forward. 4.1 expresses that an edge is either part of our route or not. 4.2 and 4.3 make sure that if (i,j) is an edge of our route then any other edge with starting point i or end point j is not part of our route. 4.4 and 4.5 express that if a vertex i is either the starting point or know to be reached and (i,j) is part of our route, then j is reached, too. Finally 4.6 says that if not all vertices are reached, then our route is not a hamilton circuit.

The starting point can be freely chosen for any $n \in \mathsf{V}$

$$T_{\mathsf{hc}}(\mathcal{G}, n) := T_{\mathsf{hc}}(\mathcal{G}) \cup \{\mathsf{initialnode}_n\}$$

We can use circumscription to verify whether there exists a hamilton circuit in a given graph $\mathcal{G}$. The variables that are to minimize are

$$P := \{\mathsf{otherroute}_{ij} : (i,j) \in \mathsf{E}\} \cup \{\mathsf{reached}_i : i \in \mathsf{V}\}\,.$$

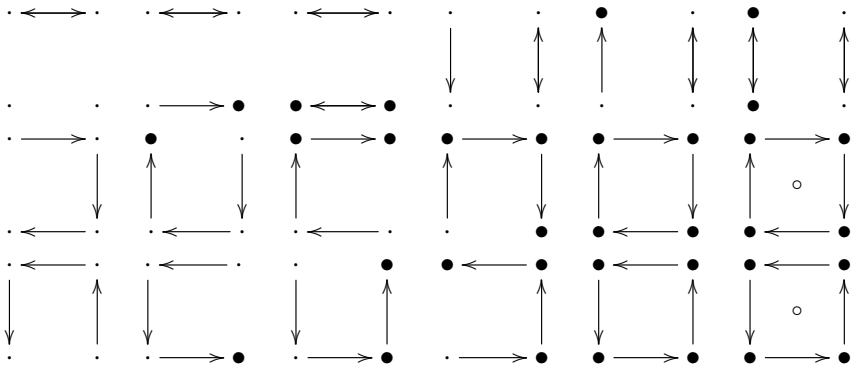
A graph $\mathcal{G}$ then has a Hamiltonian circuit with starting point n if and only if there exists a P-minimal model M of $T_{\mathsf{hc}}(\mathcal{G}, n)$ that does not contain noncircuit. To verify whether there exists a Hamiltonian circuit with starting

n	PR1u	PR2	PR2g	PR2gu	mm
2	0:00	0:00	0:00	0:00	0:00
3	0:21	0:02	0:01	0:01	0:00
4	1:06:36	0:40	0:31	0:15	0:00
5	-	14:30	11:30	4:07	0:00

Table 4.1: Proving time of $T_{\mathsf{hc}}(\mathcal{G}, n) \supset_{P(n)} \neg\mathsf{noncircuit}$

point n for a graph $\mathcal{G}$ we thus need to verify whether noncircuit is not P-minimally entailed by $T_{\mathsf{hc}}(\mathcal{G}, n)$.

We have tested our provers on $2 \times n$ grid graphs with initial node on one of its corners. For $n = 2$ the routes that correspond to minimal models are given below. We use the symbols $\bullet$ and $\cdot$ to represent reached and not reached nodes, respectively. Models that do not contain noncircuit are labeled with the symbol $\circ$ in the middle of the graph.



The proving times of the different provers are given in Table 4.1.

First of all we notice that mm is amazingly fast. In this case the prover of the LWB can not compete with it. However, the countermodels that mm finds to prove that noncircuit is not minimally entailed are not minimal. A comparison with mm is thus useless.

Let us look at the prover in the LWB. The proving strategy of PR1u is for this problem clearly disadvantageous. We look at the prover statistics for $n = 4$. PR1u encounters over 22 millions branching rules and can omit proving the second premise in only 20.231 cases. This lead to over 22 millions processed (C2) rules. Compared to that the 252 processes (C1) rules are of no consequence.

The other provers perform much better. PR2 and PR2g encounter 2.362 (C3) rules, PR2gu only 1.537. The general improvements allow PR2g to reduce the number of processed (C1) and (C2) rules from 2.898 to 2.354

and 4.717 to 2.900, respectively. Use-check allows PR2gu to omit 193 times proving the second premise. It thus ends up with 1.336 (C1) and 1.789 (C2) rules.

Compared to the previous problem, the number of processed rules is clearly smaller. Still we have longer proving times. The main case is that we no longer have easy CPC proofs or refutations, hence applying (C1) or (C2) backwards is more time intensive. This points out that reducing the number of branchings is of great importance.

4.3.7 Conclusions

Eiter and Gottlob [11] have shown that propositional circumscription is Π_2^P-complete. For worst-case problems it was thus not to expect that our improvements would lead to a major difference in proving times. Nevertheless, compared to the naive approach we were able to reduce proving time for selected problems to a certain degree.

The first two examples have shown that the approach to use the classical prover together with the circumscription theory $\mathrm{CIRC}(\Gamma, P, Q)$ is only reasonable if there are few minimize and fixed variables. The same holds for PR1.

The prover PR1u leaves an ambivalent mark. For some problems it performs amazingly fast. However, its performance depends highly on the choice of the principal formulas in the CPC axioms and also on the order in which the fixed and minimized variables are processed. The prover can show a heavy drop in performance if these two factors don't fit. Unless there is a way to influence these factors positively, PR1u is in our view not suited as a stand-alone prover in the LWB. It is however a good candidate if several proving strategies are executed simultaneously.

The provers that prefer non-branching to branching rules are less influenced by the order in which the minimized and fixed variables are processed and by the choice of the principal formulas in the CPC axioms. This robustness is sometimes paid with a certain overhead of backward applied non-branching rules. However, we have in such cases only a moderate drop in performance. Therefore we have chosen PR2gu as the standard prover for circumscription in the LWB.

Chapter 5

Propositional Default Logic

In this chapter we give the definition of propositional default logic together with some well known examples. Then we recall the sequent calculi for credulous and skeptical propositional default logic developed by Bonatti and Olivetti [4] and give some examples of sequents derived in those calculi.

For convenience we show the proofs for soundness and completeness of the calculi according to Bonatti and Olivetti. To show completeness of the residue calculus we use a slightly different approach.

5.1 Definition of Default Logic

Default logic was introduced 1980 by Reiter [24] to formalize default assumptions. It is based on the notion of a default rule which expresses that something is true by default, unless it is known to be an exception. The central definition in default logic is that of an extension of a default theory. Reiter defines it as a least fixed point. Besides this, he gives an equivalent, semi-iterative characterization of an extension, which is of more use for our work.

A propositional default rule resembles a classical deduction rule with the difference that besides a prerequisite and a consequent it contains an additional set of justifications which must be met by the end result (the extension) in order that the default can be applied.

Definition 5.1 (propositional default rule)
We define a propositional *default rule* δ to be a triple $\langle A, \{B_1, \ldots, B_n\}, C\rangle$ denoted by

$$\frac{A : B_1, \ldots, B_n}{C} \qquad \text{or} \qquad A : B_1, \ldots, B_n / C$$

where A is called the *prerequisite*, $\{B_1, \ldots, B_n\}$ the *justification* and C the *consequent* of δ.

Given a default rule δ we write $\text{pre}(\delta)$ for the prerequisite, $\text{jus}(\delta)$ for the justification and $\text{con}(\delta)$ for the consequent of it.

For a set of default rules D we write $\text{pre}(D)$, $\text{jus}(D)$ and $\text{con}(D)$ to denote the set of prerequisites, justifications and consequents of all elements of D.

We often speak of a *default* instead of a default rule.

Definition 5.2 (special default rules)
A propositional default with an empty justification is called a *residue*. Otherwise we call it a *proper default*. We often omit the colon and write A/B or $\frac{A}{B}$ to denote a residue.

A propositional default δ is called *normal* if $\text{jus}(\delta) = \{\text{con}(\delta)\}$, *semi-normal* if $\text{jus}(\delta) \Vdash \text{con}(\delta)$, and *categorical* or *prerequisite-free* if $\text{pre}(\delta)$ is empty or a tautology. For categorical defaults we often omit the prerequisite in the notation, e.g. $: B_1, \ldots, B_n/C$.

That is $\frac{A:}{B}$ is a residue, $\frac{A:B}{B}$ and $\frac{\top:A}{A}$ are normal defaults, $\frac{A:B \wedge C}{B \vee C}$ and $\frac{\top:A \leftrightarrow B}{B \to A}$ are semi-normal defaults, and $\frac{:A}{B}$ and $\frac{A \wedge B \to A \vee B : C}{D}$ are categorical or prerequisite-free defaults.
It is clear that every normal default is also semi-normal and that a normal or semi-normal default can also be categorical.

A default theory consists of a set of propositional formulas and a set of default rules. Intuitively the two sets represent some incomplete facts about a world and a set of beliefs. The latter is used to further extend the incomplete given facts. The intuitive meaning of an extension of a default theory is then a maximal extension of the given facts. Formally an extension of a default theory is defined to be the least fixed point of a certain operator Γ.

Definition 5.3 (default theory, residue theory)
A *default theory* over a language of propositional logic $\mathcal{L}(\mathcal{V})$ is a pair $\langle W, D\rangle$ consisting of a set of formulas $W \subseteq \mathcal{L}$ called the *background theory* and a set of default rules D.

A default theory $\langle W, D\rangle$ is called *normal*, *semi-normal* or *categorical* if the elements of D are normal, semi-normal or categorical, respectively.

We speak of a *residue theory* if all elements of D are residues.

Definition 5.4 (extension)
Let $\langle W, D\rangle$ be a default theory over $\mathcal{L}(\mathcal{V})$. For any set of formulas $S \subseteq \mathcal{L}(\mathcal{V})$ let $\Psi(S)$ be the smallest set of formulas satisfying the following properties:

1. $\Psi(S) = \mathrm{Th}(\Psi(S))$.
2. $W \subseteq \Psi(S)$.
3. If $A : B_1, \ldots, B_n / C \in D$, $A \in \Psi(S)$ and $\neg B_1 \notin S, \ldots, \neg B_n \notin S$, then $C \in \Psi(S)$.

A set of formulas $E \subseteq \mathcal{L}$ is called an *extension* of $\langle W, D \rangle$ if and only if $E = \Psi(E)$. We write $\mathrm{Ext}(W, D)$ for the set of extensions of $\langle W, \Delta \rangle$.

A default theory $\langle W, D \rangle$ may have an arbitrary number of extensions. Unfortunately there are default theories that have no extension at all, as an example below shows. This is due to point 3 of the definition which demands that the consequent of a default must be in an extension E if E contains its prerequisites and the justifications are consistent with the E. However, adding the consequent may invalidate justifications of previously applied defaults, therefore the existence of an extension is not guaranteed.

Reiter [24] shows that if a default theory contains only normal defaults, then an extension is guaranteed to exists. He mentions that he knows of no naturally occurring default which can not be represented in normal form. Therefore being limited to normal default theories should not be much of a restriction.

Besides the above definition of an extension, Reiter gives an equivalent, semi-iterative characterization of a extension. This theorem is interesting from an algorithmic point of view since it shows us how to compute an extension.

To compute an extension E we start with the background theory E_0 and extend it step by step. In each step we add the consequent of those defaults to E_{i+1} whose prerequisites are entailed by E_i and whose justifications are all consistent with the end result E.

The computation is called semi-iterative because the justifications must be consistent with the result E and backtracking is thus necessary.

Theorem 5.5 (extension construction)
A set of formulas $E \subseteq \mathcal{L}$ is an extension of the default theory $\langle W, D \rangle$ if and only if $E = \bigcup_{i=0}^{\infty} E_i$, where

$$E_0 := W$$
$$E_{i+1} := \mathrm{Th}(E_i) \cup \{\mathrm{con}(\delta) : \delta \in D, \mathrm{pre}(\delta) \in \mathrm{Th}(E_i) \text{ and } E \cap \neg\mathrm{jus}(\delta) = \emptyset\}$$

Proof. We quote the proof as given in the paper of Reiter [24].

We begin by observing that $\bigcup_{i=0}^{\infty} E_i$ enjoys the following properties:

D1'. $W \subseteq \bigcup_{i=0}^{\infty} E_i$

D2'. $\mathrm{Th}(\bigcup_{i=0}^{\infty} E_i) = \bigcup_{i=0}^{\infty} E_i$

D3'. If $\frac{A : B_1, \ldots, B_m}{C} \in D$ and $A \in \bigcup_{i=0}^{\infty} E_i$ and $\neg B_1, \ldots, \neg B_n \notin E$, then $C \in \bigcup_{i=0}^{\infty} E_i$.

Hence by the minimality of $\Psi(E)$, we have $\Psi(E) \subseteq \bigcup_{i=0}^{\infty} E_i$ (1).

($\Rightarrow$) We inductively prove $E_i \subseteq E$ for all $i \geq 0$, whence $\bigcup_{i=0}^{\infty} E_i \subseteq E$. Clearly, since $E = \Psi(E)$, $E_0 \subseteq E$. Assume $E_i \subseteq E$ and consider $C \in E_{i+1}$. If $C \in \mathrm{Th}(E_i)$, then since $E_i \subseteq E$ we have $C \in \mathrm{Th}(E) = E$. Otherwise there is a default $A : B_1, \ldots B_n / C \in D$, where $A \in E_i$ and $\neg B_1, \ldots, \neg B_n \notin E$. Then since $E_i \subseteq E$, $A \in E = \Psi(E)$. Hence $C \in \Psi(E)$ by point 3 of Definition 5.4 and since $\Psi(E) = E$, we have $C \in E$.

Thus $\bigcup_{i=0}^{\infty} E_i \subseteq E$. By (1) and the fact that $E = \Psi(E)$ we have $E = \bigcup_{i=0}^{\infty} E_i$.

($\Leftarrow$) We inductively prove $E_i \subseteq \Psi(E)$ for all $i \geq 0$, whence $E = \bigcup_{i=0}^{\infty} E_i \subseteq \Psi(E)$. By invoking (1) we will then have $E = \Psi(E)$ whence E is an extension of $\langle W, D \rangle$.

Clearly $E_0 \subseteq \Psi(E)$, so assume $E_i \subseteq \Psi(E)$ and consider $C \in E_{i+1}$. If $C \in \mathrm{Th}(E_i)$, then since $E_i \subseteq \Psi(E)$ we have $C \in \mathrm{Th}(\Psi(E)) = \Psi(E)$. Otherwise there is a default $A : B_1, \ldots B_n / C \in D$, where $A \in E_i$ and $\neg B_1, \ldots, \neg B_n \notin E$. Then since $E_i \subseteq \Psi(E)$, $A \in \Psi(E)$. Hence $C \in \Psi(E)$ by point 3 of Definition 5.4. Hence $E_{i+1} \subseteq \Psi(E)$. □

Reiter also shows that an extension is the closure of the background theory together with the consequents of its generating defaults.

Definition 5.6 (generating default)
A default δ is called *generating* for a set of formulas $E \subseteq \mathcal{L}$ if $\mathrm{pre}(\delta) \in E$ and $\neg \mathrm{jus}(\delta) \cap E = \emptyset$.

For a set of defaults D we define the set $\mathrm{GD}(D, E)$ of generating defaults of D for E accordingly.

$$\mathrm{GD}(D, E) := \{\delta \in D : \delta \text{ is generating for } E\}.$$

Theorem 5.7 (generating defaults as extension)
Suppose E is an extension of a default theory $\langle W, D \rangle$. Then

$$E = \mathrm{Th}(W \cup \mathrm{con}(\mathrm{GD}(D, E)))$$

For a proof of the theorem we again refer to the paper of Reiter [24].

5.1.1 Default Entailment

As a circumscription theory may have an arbitrary number of minimal models, a default theory may have an arbitrary number of extensions. In the original paper about default reasoning Reiter defined a formula A to be valid in a default theory $\langle W, D\rangle$ if A was contained in an extension of $\langle W, D\rangle$. Nowadays two kind of entailment have been established for default logic, a skeptical and a credulous one.

Definition 5.8 (skeptical and credulous entailment)
A default theory $\langle W, D\rangle$ *skeptically entails* a formula A if A is contained in every extension of $\langle W, D\rangle$.

A default theory $\langle W, D\rangle$ *credulously entails* a formula A if A is contained in an extension of $\langle W, D\rangle$.

5.1.2 Examples

Nixon Diamond

The first example we present is called Nixon Diamond and was introduced by Reiter and Criscuolo [25]. It is about Richard Nixon[1] who is a republic and a quaker. Now usually quakers are pacifists while republics are not. The situation is modeled with the following default theory.

$$\langle W, D\rangle := \left\langle \{q, r\}, \left\{ \frac{r : \neg p}{\neg p}, \frac{q : p}{p} \right\} \right\rangle$$

The background theory contains the facts that Nixon is a quaker (q) and a republic (r). The default rules express that "if Nixon is a republic then, if possible, assume that he is not a pacifist" and "if Nixon is a quaker then, if possible, assume that he is a pacifist".

It is easy to verify that $\langle W, D\rangle$ has two extensions: $E_1 := \mathrm{Th}(\{q, r, p\})$ and $E_2 := \mathrm{Th}(\{q, r, \neg p\})$. This is because if the first default rule is applied then this excludes the second from being applied and vice versa.

In skeptical entailment, neither p nor $\neg p$ would be deducible while in the credulous case both formulas could be derived.

[1] The original example was not about Richard Nixon but about John.

Default theory without extension

The second example is about a default theory without extension.

$$\langle W, D\rangle := \left\langle \emptyset, \left\{ \frac{:p}{\neg p}, \right\} \right\rangle$$

It is easy to see that this theory has no extension. Suppose there exists an extension E of this theory. Then $\delta := :p/\neg p$ is either generating for E or not. Suppose it is, then $\neg p \in E = \mathrm{Th}(\{\neg p\})$ (Theorem 5.7) but then δ is not generating for E since $\neg\mathrm{jus}(\delta) \cap E = \{\neg p\}$ which is a contradiction to our supposition. Thus δ is not generating for E. Then $\mathrm{GD}(D, E) = \emptyset$, i.e. $E = \mathrm{Th}(\emptyset)$ (Theorem 5.7). But then $:p/\neg p$ is generating for E, which contradicts our supposition again.

In skeptical entailment this default theory would entail any formula. In the credulous case however, for lack of a witness, nothing could be entailed.

5.2 Sequent Calculi for Default Logic

In this section we recall the sequent calculi for propositional default logic developed by Bonatti and Olivetti [4]. There are two variants, one for skeptical and one for credulous default logic. Both variants make use of the so called propositional residue calculus, a calculus used for residue theories. We first introduce that calculus and then continue with the definition of the credulous and skeptical propositional default calculus.

5.2.1 Preliminaries

For residue and default sequents we use the enriched languages $\mathcal{L}^{res}$ and $\mathcal{L}^{def}$.

Definition 5.9 ($\mathcal{L}^{res}, \mathcal{L}^{def}$)
We define the language of propositional residue logic $\mathcal{L}^{res}$ to consists of the language of propositional logic and the set of propositional residues.

$$\mathcal{L}^{res} := \mathcal{L} \cup \left\{ \frac{A}{C} : A \in \mathcal{L}, C \in \mathcal{L} \right\}$$

The language of propositional default logic $\mathcal{L}^{def}$ is defined accordingly.

$$\mathcal{L}^{def} := \mathcal{L} \cup \left\{ \frac{A : B_1, \ldots, B_n}{C} : A \in \mathcal{L}, B_i \in \mathcal{L} \text{ for } 1 \leq i \leq n, C \in \mathcal{L} \right\}$$

5.2.2 Propositional Residue Calculus

The calculi for skeptical and credulous default logic presented later are based on the propositional residue calculus PRC and the propositional residue refutation calculus PRRC[2]. Residue theories have the property that they have exactly one extension which can be calculated iteratively.
In this section we first introduce the closure operator on residue theories and give some useful properties that are needed for the main result. Then we give two lemmas that show how default rules can be eliminated or simplified to residues. These lemmas are later used to show soundness and completeness of the default calculi. We close the section with the introduction of PRC and PRRC and show that both calculi are sound and complete.

Definition 5.10 ($\mathrm{Res}(D, E)$**)**
Given a set of default rules D and a set of formulas $E \subseteq \mathcal{L}$ we define

$$\mathrm{Res}(D, E) := \{\mathrm{pre}(\delta)/\mathrm{con}(\delta) : \delta \in D \text{ and } E \cap \neg\mathrm{jus}(\delta) = \emptyset\}$$

The definition $\mathrm{Res}(D, E)$ is similar to the one $\mathrm{GD}(D, E)$. The differences are that $\mathrm{Res}(D, E)$ returns a set of residues by cutting off the justifications of the defaults and that the condition that $\mathrm{pre}(\delta) \in E$ must not be met, i.e. $\{\delta \in \mathrm{Res}(D, E) : \mathrm{pre}(\delta) \in E\} = \{\mathrm{pre}(\delta)/\mathrm{con}(\delta) : \delta \in \mathrm{GD}(D, E)\}$.

Definition 5.11 (closure $\mathrm{Cl}(W, R)$**)**
Given a set of formulas $W \subset \mathcal{L}$ and a set of residues R we define the closure $\mathrm{Cl}(W, R)$ as follows:

$$\mathrm{Cl}(W, R) := \bigcup_{i=0}^{\infty} \mathrm{Cl}_i(W, R)$$

where

$$\begin{aligned} \mathrm{Cl}_0(W, R) &:= W \\ \mathrm{Cl}_{i+1}(W, R) &:= \mathrm{Th}(\mathrm{Cl}_i(W, R)) \cup \{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}_i(W, R))\}. \end{aligned}$$

The similarity between the definition of the closure operator and the semi-iterative definition of an extension is obvious. If E is an extension of the default theory $\langle W, D\rangle$ then it is easy to see that $E_i = \mathrm{Cl}_i(W, \mathrm{Res}(D, E))$. This leads immediately to the following proposition.

Proposition 5.12 (closure as extension)
E is an extension of $\langle W, D\rangle$ if and only if $E = \mathrm{Cl}(W, \mathrm{Res}(D, E))$.

The proof of this proposition can be found in the paper of Bonatti and Olivetti [4].

[2] Olivetti and Bonatti [4] define only one calculus but make use of so called anti-sequents such that proving and refuting a sequent is done in the same calculus. To us it seemed more natural to introduce two calculi, one for proving a sequent an one for refuting it.

The following lemma states some important properties of the closure operator. More precisely, that it is monotonic with respect to both parameters, contains the background theory, is closed under classical entailment and contains the consequent of a residue if it contains its prerequisite.

Lemma 5.13 (monotony of $\mathrm{Cl}(W, R)$)
Let $\langle W, R\rangle$ and $\langle W', R'\rangle$ be residue theories. Then the following holds.

1. If $W \subseteq W'$ and $R \subseteq R'$, then $\mathrm{Cl}(W, R) \subseteq \mathrm{Cl}(W', R')$.
2. $W \subseteq \mathrm{Cl}(W, R)$.
3. $\mathrm{Th}(\mathrm{Cl}(W, R)) \subseteq \mathrm{Cl}(W, R)$.
4. If $A \in \mathrm{Cl}(W, R \cup \{A/B\})$, then $B \in \mathrm{Cl}(W, R \cup \{A/B\})$.

The proof of this lemma can be found in the paper of Bonatti and Olivetti [4].

The following lemma is of use to show completeness and soundness of PRC and PRRC. It is according to a lemma in the paper of Bonatti and Olivetti but has been extended with an additional claim which we use to proof completeness of PRC and PRRC.

Lemma 5.14 (properties of $\mathrm{Cl}(W, R)$)
For a residue theory $\langle W, R\rangle$ the following holds.

1. If $A \in \mathrm{Cl}(W, R)$, then $\mathrm{Cl}(W, R \cup \{A/B\}) = \mathrm{Cl}(W \cup \{B\}, R)$.
2. If $A \notin \mathrm{Cl}(W, R)$, then $\mathrm{Cl}(W, R \cup \{A/B\}) = \mathrm{Cl}(W, R)$.
3. $\mathrm{Cl}(W, R \cup \{A/B\}) \subseteq \mathrm{Cl}(W \cup \{B\}, R)$.
4. If $A \in \mathrm{Cl}(W, R \cup \{A/C\})$ then $A \in \mathrm{Cl}(W, R)$

Proof. For the proofs of claim 1, 2 and 3 we refer to the paper of Olivetti and Bonatti [4]. We only show claim 4.
Suppose $A \in \mathrm{Cl}(W, R \cup \{A/C\})$ but $A \notin \mathrm{Cl}(W, R)$. Then according to 2. $\mathrm{Cl}(W, R) = \mathrm{Cl}(W, R \cup \{A/C\}) \not\ni A$ which contradicts to our supposition. □

The default calculi presented later make use of constraints that are used to deduce defaults from residues. In the two lemmas that follow we show how to eliminate default rules or simplify them to residues. These lemmas are later used to show soundness and completeness of the default calculi.

Lemma 5.15 (eliminating defaults)
Let $E \subseteq \mathcal{L}$. If $E \cap \neg\mathrm{jus}(\delta) \neq \emptyset$, then E is an extension of $\langle W, D \cup \{\delta\}\rangle$ if and only if E is an extension of $\langle W, D\rangle$.

$$\frac{\Gamma \supset \Delta}{\Gamma, A/C \supset \Delta}(\text{Res1}) \qquad \frac{\Gamma \supset A \quad \Gamma, C \supset \Delta}{\Gamma, A/C \supset \Delta}(\text{Res2})$$

Figure 5.1: Residue rules of PRC

$$\frac{\Gamma \supset \Delta \quad \Gamma \supset A}{\Gamma, A/C \supset \Delta}(\text{Res3}) \qquad \frac{\Gamma, C \supset \Delta}{\Gamma, A/C \supset \Delta}(\text{Res4})$$

Figure 5.2: Residue rules of PRRC

Lemma 5.16 (defaults as residues)
Let $E \subseteq \mathcal{L}$. If $E \cap \neg\text{jus}(\delta) = \emptyset$, then E is an extension of $\langle W, D \cup \{\delta\}\rangle$ if and only if E is an extension of $\langle W, D \cup \{\text{pre}(\delta)/\text{con}(\delta)\}\rangle$.

For the proofs of the above lemmas we refer to the paper of Bonatti and Olivetti [4].

It is well known, that the default theory $\langle W, \emptyset\rangle$ has exactly one extension. For a residue theory $\langle W, R\rangle$ this also holds.

Theorem 5.17 (uniqueness of $\text{Cl}(W, R)$)
Let $\langle W, R\rangle$ be a residue theory. Then $\text{Cl}(W, R)$ is the unique extension of $\langle W, R\rangle$.

A proof of this theorem can be found in the paper of Bonatti and Olivetti [4].

The calculi for skeptical and credulous default calculi rely on provability and refutability of residue sequents. Their definition is straightforward.

Definition 5.18 (residue sequent)
A *residue sequent* is a pair $\langle \Gamma, \Delta\rangle$ denoted by $\Gamma \supset \Delta$, consisting of a finite multiset $\Gamma \subseteq \mathcal{L}^{res}$ and a finite multiset of formulas $\Delta \subseteq \mathcal{L}$.

$\Gamma \supset \Delta$ is defined to be *valid* if $\bigvee \Delta \in \text{Cl}(\Gamma \cap \mathcal{L}, \Gamma \setminus \mathcal{L})$. We denote this as usual with $\models \Gamma \supset \Delta$.

Definition 5.19 (PRC)
We define the *propositional residue calculus* PRC to have the deduction rules of CPC and those given in Figure 5.1. In both rules we call A/C the *principal residue*. The formulas A and C are called *active formulas in the premise*.

Definition 5.20 (PRRC)
We define the *propositional residue refutation calculus* PRRC to have the deduction rules of CPRC and those given in Figure 5.2. In both rules we call A/C the *principal residue*. The formulas A and C are called *active formulas in the premise*.

We say that a residue sequent is *refutable* if it is deducible in PRRC.

Remark 5.21 (ambiguity regarding residue calculi)
In the sequent calculi for credulous and skeptical default logic we rely on the provability and the refutability of a residue sequent. To distinct this we write $\mathsf{PRC} \vdash \Gamma \supset \Delta$ and $\mathsf{PRRC} \vdash \Gamma \supset \Delta$ in a deduction rule to indicate that to apply the rule a residue sequent has to be provable and refutable, respectively.

Remark 5.22 (CPC and CPRC rules in PRC and PRRC)
The rules of CPC and CPRC in PRC and PRRC, respectively, may only be applied on pure CPC sequents, i.e. sequents that do not have a residue in its antecedent.
Consider the valid residue sequent $p, p/q \supset q$. Applying $(\supset \neg)$ on it results in the invalid residue sequent $p/q \supset \neg p, q$.

The deduction rules of the two calculi reflect some lemmas defined above. (Res1) for example reflects the monotonicity of the closure operator while the rules (Res2), (Res3) and (Res4) reflect Lemma 5.14.1, 5.14.2 and 5.14.3, respectively.

Theorem 5.23 (soundness of PRC)
Let $\Gamma \supset \Delta$ be a residue sequent.

If $\Gamma \supset \Delta$ is deducible in PRC then it is valid.

Proof. Let $W := \Gamma \cap \mathcal{L}$ and $R := \Gamma \setminus \mathcal{L}$. We show our claim by induction on $|R|$.

$|R| = 0$: Then the claim follows from the soundness of CPC.

$|R| = n + 1$: Then R is of the form $R', A/C$ and the last rule in the proof was either (Res1) or (Res2).

If the last rule was (Res1) then we know by induction hypothesis that $\bigvee \Delta \in \mathrm{Cl}(W, R')$. By Lemma 5.13.1 we thus obtain $\bigvee \Delta \in \mathrm{Cl}(W, R)$.

If the last rule was (Res2) then we know by induction hypothesis that $\bigvee \Delta \in \mathrm{Cl}(W \cup \{C\}, R')$ and $A \in \mathrm{Cl}(W, R')$. With Lemma 5.14.1 we then know $\mathrm{Cl}(W, R' \cup \{A/C\}) = \mathrm{Cl}(W \cup \{C\}, R') \ni \bigvee \Delta$. □

Theorem 5.24 (completeness of PRC)
Let W be a finite set of formulas and R be a finite set of residues.

If $\bigvee \Delta \in \mathrm{Cl}(W, R)$ then $W, R \supset \Delta$ is deducible in PRC.

Proof. We show our claim by induction on $|R|$.

Suppose $\bigvee \Delta \in \mathrm{Cl}(W, R)$.

$|R| = 0$: Then the claim follows from the completeness of CPC.

$|R| = n + 1$: Let $A/C \in R$ and $R' := R \setminus \{A/C\}$.
We distinguish two cases.

$A \in \mathrm{Cl}(W, R)$: Then according to Lemma 5.14.4 $A \in \mathrm{Cl}(W, R')$. Furthermore by Lemma 5.14.3 we know $\mathrm{Cl}(W \cup \{C\}, R') \supseteq \mathrm{Cl}(W, R)$. Thus by induction hypothesis $W, R' \supset A$ and $W, C, R' \supset \Delta$ are deducible in PRC. With (Res2) we can deduce $W, R \supset \Delta$.

$A \notin \mathrm{Cl}(W, R)$: Then by Lemma 5.13.1 we know $A \notin \mathrm{Cl}(W, R')$. With Lemma 5.14.2 we obtain $\mathrm{Cl}(W, R') = \mathrm{Cl}(W, R)$. Thus $W, R' \supset \Delta$ is deducible in PRC by induction hypothesis. Using (Res1) we can derive $W, R \supset \Delta$. □

Theorem 5.25 (soundness of PRRC)
Let $\Gamma \supset \Delta$ be a residue sequent.

If $\Gamma \supset \Delta$ is refutable then it is not valid.

Proof. Let $W := \Gamma \cap \mathcal{L}$ and $R := \Gamma \setminus \mathcal{L}$. We show our claim by induction on $|R|$.

$|R| = 0$: Then $\mathrm{Cl}(W, R) = \mathrm{Th}(W)$ and the claim follows from the soundness of CPRC.

$|R| = n + 1$: Then R is of the form $R', A/C$ and the last rule in the proof was either (Res3) or (Res4).

If the last rule was (Res3) then we know by induction hypothesis that $\bigvee \Delta \notin \mathrm{Cl}(W, R')$ and $A \notin \mathrm{Cl}(W, R')$. With Lemma 5.14.2 we then know $\mathrm{Cl}(W, R' \cup \{A/C\}) = \mathrm{Cl}(W, R') \not\ni \bigvee \Delta$.

If the last rule was (Res4) then we know $\bigvee \Delta \notin \mathrm{Cl}(W \cup \{C\}, R')$ by induction hypothesis. With Lemma 5.14.3 we obtain $\mathrm{Cl}(W, R' \cup \{A/C\}) \subseteq \mathrm{Cl}(W \cup \{C\}, R') \not\ni \bigvee \Delta$. □

Theorem 5.26 (completeness of PRRC)
Let W be a finite set of formulas and R be a finite set of residues.

If $\bigvee \Delta \notin \mathrm{Cl}(W, R)$ then $W, R \supset \Delta$ is refutable.

Proof. We show our claim by induction on $|R|$.

Suppose $\bigvee \Delta \notin \mathrm{Cl}(W, R)$.

$|R| = 0$: Then the sequent is refutable since CPRC is complete.

$|R| = n + 1$: Let $A/C \in R$ and $R' := R \setminus \{A/C\}$. We distinguish two cases.

$A \in \mathrm{Cl}(W, R)$: Then according to Lemma 5.14.4 $A \in \mathrm{Cl}(W, R')$. With Lemma 5.14.1 we obtain $\mathrm{Cl}(W, R) = \mathrm{Cl}(W \cup \{C\}, R')$. Thus $W, C, R' \supset \Delta$ is refutable by induction hypothesis. With (Res4) we can deduce $W, R \supset \Delta$.

$A \notin \mathrm{Cl}(W, R)$: With Lemma 5.13.1 we then know $A \notin \mathrm{Cl}(W, R')$ and by Lemma 5.14.2 we obtain $\mathrm{Cl}(W, R') = \mathrm{Cl}(W, R)$. Therefore $W, R' \supset A$ and $W, R' \supset \Delta$ are refutable by induction hypothesis. Using (Res3) we can derive $W, R \supset \Delta$. □

5.2.3 Credulous Propositional Default Calculus

Now we present the sequent calculus cPDC of Bonatti and Olivetti [4] which we use for proof search in credulous propositional default logic. The term "credulous" is according to Definition 5.8 on page 113 which defines a default theory $\langle W, D\rangle$ to credulously entail a formula A if A is contained in at least one extension of $\langle W, D\rangle$.
We start by defining the kind of sequents the calculus makes use of, continue with the definition of cPDC and close the section with the proofs of soundness and completeness of cPDC.
The definitions and proofs in this section are all according to Bonatti and Olivetti [4].

The calculi for propositional default logic make use of constraints on extension. We use an additional operator to distinguish them from normal formulas.

Definition 5.27 (constraints)
A *constraint* is a formula of the form $\mathbf{L}A$ or $\neg\mathbf{L}A$, where $A \in \mathcal{L}$.

$\mathbf{L}A$ is fulfilled by a set of formulas Γ if $\Gamma \Vdash A$. Accordingly $\neg\mathbf{L}A$ is fulfilled by Γ if $\Gamma \nVdash A$.
A set Σ of constraints is fulfilled by Γ if it fulfills all elements of Σ.

Definition 5.28 (default sequent)
A *default sequent* is a triple $\langle \Sigma, \Gamma, \Delta\rangle$ denoted by $\Sigma; \Gamma \supset \Delta$ where Σ is a finite multiset of provability constraints, Γ is a finite multiset containing a default theory (consisting of formulas and default rules) and Δ is a finite multiset of formulas.

Definition 5.29 (credulously valid default sequent)
A default sequent $\Sigma; \Gamma \supset \Delta$ is defined to be *credulously valid* if there exists an extension E of Γ such that E fulfills Σ and $\bigvee \Delta \in E$. We then call E a *witness* for $\Sigma; \Gamma \supset \Delta$.

Definition 5.30 (cPDC)
We define the credulous propositional default calculus cPDC to have the deduction rules given in Figure 5.3.

The deduction rules (cD1), (cD2) and (cD3) can be regarded as interfacing rules from residue sequents to default sequents. (cD4) and (cD5) reflect Lemma 5.16 and 5.15, respectively.

$$\frac{\mathsf{PRC} \vdash \Gamma \supset \Delta}{;\Gamma \supset \Delta}(\mathsf{cD1})^*$$

$$\frac{\mathsf{PRC} \vdash \Gamma \supset A \quad \Sigma;\Gamma \supset \Delta}{\Sigma, \mathbf{L}A;\Gamma \supset \Delta}(\mathsf{cD2})^* \qquad \frac{\mathsf{PRRC} \vdash \Gamma \supset A \quad \Sigma;\Gamma \supset \Delta}{\Sigma, \neg\mathbf{L}A;\Gamma \supset \Delta}(\mathsf{cD3})^*$$

$$\frac{\Sigma, \mathbf{L}\neg B_i;\Gamma \supset \Delta}{\Sigma;\Gamma, \dfrac{A : B_1, \ldots, B_n}{C} \supset \Delta}(\mathsf{cD4}) \qquad \frac{\Sigma, \neg\mathbf{L}\neg B_1, \ldots, \neg\mathbf{L}\neg B_n;\Gamma, \dfrac{A}{C} \supset \Delta}{\Sigma;\Gamma, \dfrac{A : B_1, \ldots, B_n}{C} \supset \Delta}(\mathsf{cD5})$$

$^*\ \Gamma \subseteq \mathcal{L}^{res}$

Figure 5.3: Default deduction rules of cPDC

Theorem 5.31 (soundness of cPDC)
If a default sequent is deducible in cPDC then it is credulously valid.

Proof. Since PRC and PRRC are sound it is sufficient to show for the rules (cD1) – (cD5) that if all their premises are credulously valid then so is their conclusion. In the following let $W := \Gamma \cap \mathcal{L}$ and $D := \Gamma \setminus \mathcal{L}$.

(cD1)**:** Suppose that $\Gamma \supset \Delta$ is valid (1) and let $E := \mathrm{Cl}(W, D)$ be the unique extension of Γ. From (1) we know $\bigvee \Delta \in E$. Hence the conclusion is credulously valid.

(cD2)**:** Suppose that $\Gamma \supset A$ is valid (1), $\Sigma;\Gamma \supset \Delta$ is credulously valid (2) and let $E := \mathrm{Cl}(W, D)$ be the unique extension of Γ. From (2) we know that $\bigvee \Delta \in E$ and that E fulfills Σ. From (1) we furthermore know that E also fulfills $\mathbf{L}A$. Thus $\Sigma, \mathbf{L}A;\Gamma \supset \Delta$ is valid, too.

(cD3)**:** Suppose that $\Gamma \supset A$ is not valid (1), $\Sigma;\Gamma \supset \Delta$ is valid (2) and let $E := \mathrm{Cl}(W, D)$ be the unique extension of Γ. From (2) we know that $\bigvee \Delta \in E$ and that E fulfills Σ. From (1) we furthermore know that E also fulfills $\neg\mathbf{L}A$. Thus $\Sigma, \neg\mathbf{L}A;\Gamma \supset \Delta$ is credulously valid, too.

(cD4)**:** Suppose $\Sigma, \mathbf{L}\neg B_i;\Gamma \supset \Delta$ is credulously valid. Then there exists a witness E for it. Since E fulfills $\mathbf{L}\neg B_i$ we know that $\neg B_i \in E \cap \{\neg B_1, \ldots, \neg B_n\}$. Thus, by Lemma 5.15, E is also an extension of $\Gamma \cup \{A : B_1, \ldots, B_n/C\}$ and therefore also a witness for the conclusion $\Sigma;\Gamma, A : B_1, \ldots, B_n/C \supset \Delta$.

(cD5)**:** Suppose $\Sigma, \neg\mathbf{L}\neg B_1, \ldots, \neg\mathbf{L}\neg B_n;\Gamma, A/C \supset \Delta$ is credulously valid. Then there exists a witness E for it. Since E fulfills $\neg\mathbf{L}\neg B_i$ for all $1 \leq i \leq n$ we know that $E \cap \{\neg B_1, \ldots, \neg B_n\} = \emptyset$. Thus, by Lemma

5.16, E is also an extension of $\Gamma \cup \{A : B_1, \ldots, B_n/C\}$ and therefore also a witness for the conclusion $\Sigma; \Gamma, A : B_1, \ldots, B_n/C \supset \Delta$.

□

Theorem 5.32 (completeness of cPDC)
If a default sequent $\Sigma; \Gamma \supset \Delta$ is credulously valid then it is deducible in cPDC.

Proof. Let $W := \Gamma \cap \mathcal{L}$ and $D := \Gamma \setminus \mathcal{L}$ and suppose that $\Sigma; W, D \supset \Delta$ is credulously valid and E a witness for it. We show our claim by induction on the number d of proper defaults of D, i.e. $d := |\{\delta : \delta \in D \text{ and } \mathrm{jus}(\delta) \neq \emptyset\}|$.

- $d = 0$: Then $E := \mathrm{Cl}(W, D)$.
  We proceed by a nested induction on $|\Sigma|$.
  - $|\Sigma| = 0$: Since PRC is complete and $\bigvee \Delta \in E$ we can derive $W, D \supset \Delta$ in PRC. Using (cD1) we can derive $; W, D \supset \Delta$.
  - $|\Sigma| > 0$: Let σ be an arbitrary element of Σ. If σ is of the form $\mathbf{L}A$ [$\neg\mathbf{L}A$] then $A \in \mathrm{Cl}(W, D)$ [$A \notin \mathrm{Cl}(W, D)$]. Thus $W, D \supset A$ is deducible in PRC [PRRC]. Furthermore E is also a witness for $\Sigma \setminus \{\sigma\}; W, D \supset \Delta$ which is deducible by (nested) induction hypothesis. Using (cD2) [(cD3)] we can thus derive $\Sigma; W, D \supset \Delta$.
- $d > 0$: Let $A : B_1, \ldots, B_n/C$ be an arbitrary element of D and let $D' := D \setminus \{A : B_1, \ldots, B_n/C\}$. We distinguish two cases:
  - $E \cap \{B_1, \ldots, B_n\} \neq \emptyset$.
    Then E fulfills $\mathbf{L}\neg B_i$ for some $1 \leq i \leq n$. Furthermore, by Lemma 5.15, E is an extension of $\langle W, D' \rangle$. Thus $\Sigma, \mathbf{L}\neg B_i; W, D' \supset \Delta$ is deducible in cPDC by induction hypothesis. Using (cD4) we can derive $\Sigma; W, D \supset \Delta$ from it.
  - $E \cap \{B_1, \ldots, B_n\} = \emptyset$.
    Then E fulfills $\neg\mathbf{L}\neg B_i$ for all $1 \leq i \leq n$. Furthermore, by Lemma 5.16, E is an extension of $\langle W, D' \cup \{A/C\} \rangle$. By induction hypothesis $\Sigma, \neg\mathbf{L}\neg B_1, \ldots, \neg\mathbf{L}\neg B_n; W, D', A/C \supset \Delta$ is thus deducible in cPDC. Using (cD5) we can derive $\Sigma; W, D \supset \Delta$ from it.

□

5.2.4 Skeptical Propositional Default Calculus

We now present the sequent calculus sPDC of Bonatti and Olivetti [3] which we use for proof search in skeptical propositional default logic. The term "skeptical" is according to Definition 5.8 on page 113 which defines a default

$$\frac{\mathsf{PRC} \vdash \Gamma \supset \Delta}{\Sigma; \Gamma \supset \Delta}(\mathsf{sD1})^*$$

$$\frac{\mathsf{PRC} \vdash \Gamma \supset A}{\Sigma, \neg\mathbf{L}A; \Gamma \supset \Delta}(\mathsf{sD2})^* \qquad \frac{\mathsf{PRRC} \vdash \Gamma \supset A}{\Sigma, \mathbf{L}A; \Gamma \supset \Delta}(\mathsf{sD3})^*$$

$$\frac{\Sigma, \neg\mathbf{L}\neg B_1, \ldots, \neg\mathbf{L}\neg B_n; \Gamma, \dfrac{A}{C} \supset \Delta \quad \Sigma, \mathbf{L}\neg B_1; \Gamma \supset \Delta \ \ldots \ \Sigma, \mathbf{L}\neg B_n; \Gamma \supset \Delta}{\Sigma; \Gamma, \dfrac{A : B_1, \ldots, B_n}{C} \supset \Delta}(\mathsf{sD4})$$

* $\Gamma \subseteq \mathcal{L}^{res}$

Figure 5.4: Default deduction rules of sPDC

theory $\langle W, D\rangle$ to skeptically entail a formula A if A is contained in every extension of $\langle W, D\rangle$.
We first give the definition of the sPDC and then show soundness and completeness of it.
The definitions and proofs in this section are all according to Bonatti and Olivetti [3].

Definition 5.33 (skeptically valid default sequent)
A default sequent $\Sigma; \Gamma \supset \Delta$ is defined to be *skeptically valid* if $\bigvee \Delta \in E$ for every extension E of Γ that fulfills Σ.

Definition 5.34 (sPDC)
We define the skeptical propositional default calculus sPDC to have the deduction rules of PRC, PRRC and those given in Figure 5.4.

The deduction rules (sD1), (sD2) and (sD3) can be regarded as interfacing rules from residue sequents to default sequents. In contrast to the credulous case the constraints are not built up element-wise by those rules. On the one hand we have the rule (sD1) which aims at a default sequent whose succedent is in the extension of its default theory and thus (for any set of constraints) valid, on the other hand we have the rules (sD2) and (sD3) that aim at default sequents whose constraints are not fulfilled by any extension of their default theory.
The single non-interfacing deduction rule (sD4) reflects Lemmas 5.15 and Lemma 5.16 together.

Theorem 5.35 (soundness of sPDC)
If a default sequent is deducible in sPDC then it is skeptically valid.

Proof. Since PRC and PRRC are sound it is sufficient to show soundness for the rules (sD1) – (sD4) that if their premises are (skeptically) valid then so is their conclusion. In the following let $W := \Gamma \cap \mathcal{L}$ and $D := \Gamma \setminus \mathcal{L}$.

(sD1): Suppose that $\Gamma \supset \Delta$ is valid (1) and let $E := \mathrm{Cl}(W, D)$ be the unique extension of W, D. From (1) we know $\bigvee \Delta \in E$. Therefore the conclusion is skeptically valid, independent of whether Σ is fulfilled by E or not.

(sD2): Suppose that $\Gamma \supset A$ is valid (1) and let $E := \mathrm{Cl}(W, D)$ be the unique extension of Γ. From (1) we know that $A \in E$. Therefore E fulfills $\neg \mathbf{L}A$ not. Since E is the only extension of Γ we thus know that $\Sigma, \neg \mathbf{L}A; \Gamma \supset \Delta$ is skeptically valid for any set of formulas $\Delta \subseteq \mathcal{L}$.

(sD3): Suppose that $\Gamma \supset A$ is not valid (1) and let $E := \mathrm{Cl}(W, D)$ be the unique extension of Γ. From (1) we know that $A \notin E$. Therefore E fulfills $\mathbf{L}A$ not. Since E is the only extension of Γ we thus know that $\Sigma, \mathbf{L}A; \Gamma \supset \Delta$ is skeptically valid for any set of formulas $\Delta \subseteq \mathcal{L}$.

(sD4): Suppose that this rule is not sound, that is, there exists a default sequent $\Sigma; \Gamma, \delta \supset \Delta$ with $\delta = A : B_1, \ldots, B_n / C$ such that

(a) $\Sigma, \neg \mathbf{L} \neg B_1, \ldots, \neg \mathbf{L} \neg B_n; \Gamma, A/C \supset \Delta$ is skeptically valid

(b) $\Sigma, \mathbf{L} \neg B_i; \Gamma \supset \Delta$ is skeptically valid for all $1 \leq i \leq n$.

(c) $\Sigma; \Gamma, \delta \supset \Delta$ is not skeptically valid.

According to (c) there exists an extension E of $\Gamma \cup \{\delta\}$ that fulfills Σ and that does not contain $\bigvee \Delta$.
Suppose that E fulfills $\neg \mathbf{L} \neg B_1, \ldots, \neg \mathbf{L} \neg B_n$, then $E \cap \neg \mathrm{jus}(\delta) = \emptyset$. Thus, by Lemma 5.16, E is also an extension of $\Gamma \cup \{A/C\}$. Since E also fulfills Σ it contains $\bigvee \Delta$ according to (a), which is a contradiction to (c).
Thus E fulfills $\mathbf{L} \neg B_i$ for some $1 \leq i \leq n$. That is $E \cap \neg \mathrm{jus}(\delta) \neq \emptyset$ and thus, by Lemma 5.16, E is also an extension of Γ. Since E fulfills Σ and $\mathbf{L} \neg B_i$ it contains $\bigvee \Delta$ according to (b), which again contradicts to (c).

□

Theorem 5.36 (completeness of sPDC)
If a default sequent $\Sigma; \Gamma \supset \Delta$ is skeptically valid then it is deducible in sPDC.

Proof. Suppose that the sequent $\mathrm{S} := \Sigma; \Gamma \supset \Delta$ is skeptically valid and let $W := \Gamma \cap \mathcal{L}$ and $D := \Gamma \setminus \mathcal{L}$. We show our claim by induction on the number d of proper defaults in D, i.e. $d := |\{\delta : \delta \in D \text{ and } \mathrm{jus}(\delta) \neq \emptyset\}|$.

- $d = 0$: Then $E := \mathrm{Cl}(W, D)$ is the unique extension of Γ. We distinguish two cases.

(a) $\bigvee \Delta \in E$. Then $\Gamma \supset \Delta$ is deducible in PRC and with (sD1) we can deduce $\Sigma; \Gamma \supset \Delta$ from it.

(b) $\bigvee \Delta \notin E$. Then $\Gamma \supset \Delta$ is not valid. Since $\Sigma; \Gamma \supset \Delta$ is skeptically valid there must thus be a constraint $\sigma \in \Sigma$ which is not fulfilled by E.
If σ is of the form $\neg \mathbf{L}A$ then A must be an element of E, thus $\Gamma \supset A$ is deducible in PRC and with (sD2) we can deduce $\Sigma; \Gamma \supset \Delta$ from it.
If σ is of the form $\mathbf{L}A$ then A is not an element of E, thus $\Gamma \supset A$ is refutable and with (sD3) we can deduce $\Sigma; \Gamma \supset \Delta$ from it.

- $d = n + 1$: Then Γ is of the form $\Gamma' \cup \delta$ with $\delta = A : B_1, \ldots, B_n / C$. Since $d > 0$ the proof of S must end with an instance of (sD4). Now suppose that S is not deducible in sPDC. Then one of the premises of the last deduction rule is not deducible in sPDC.
Suppose that $\mathrm{S}_0 := \Sigma, \neg \mathbf{L} \neg B_1, \ldots, \neg \mathbf{L} \neg B_n; \Gamma', A/C \supset \Delta$ is not deducible in sPDC and thus by induction hypothesis not skeptically valid. Therefore there exists an extension E of $\Gamma' \cup \{A/C\}$ that fulfills the constraints of S_0 but does not contain $\bigvee \Delta$. Thus $E \cap \neg \mathrm{jus}(\delta) = \emptyset$ and so by Lemma 5.16 E is also an extension of Γ', δ. Since E fulfills Σ, this contradicts to our supposition that S is skeptically valid.
Hence a premise of the form $\mathrm{S}_i := \Sigma, \mathbf{L} \neg B_i; \Gamma' \supset \Delta$ is not deducible in sPDC and thus by induction hypothesis not skeptically valid. Therefore there exists an extension E of Γ' that fulfills the constraints of S_i but does not contain $\bigvee \Delta$. Thus $\neg B_i \in E \cap \neg \mathrm{jus}(\delta) \neq \emptyset$ and so by Lemma 5.15 E is also an extension of Γ', δ. Since E fulfills Σ, this contradicts to our supposition that S is skeptically valid.

□

5.2.5 Examples

We close this chapter with two example proofs which use the two example default theories presented in a previous section.

The first proof is about the "Nixon Diamond" example and shows that the corresponding theory credulously entails that Nixon is not a pacifist. We have omitted the residue part of the proof. The proof is given in Figure 5.5.

The second proof is about the default theory without extension and shows that in such a theory an arbitrary set of formulas Δ is skeptically entailed. The proof is given in Figure 5.6.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{PRC} \vdash q, r, \dfrac{r}{\neg p} \supset \neg p}}{\mathbf{L}\neg p, \neg\mathbf{L}\neg\neg p; q, r, \dfrac{r}{\neg p} \supset \neg r}(\mathsf{cD2}) \quad \dfrac{\dfrac{\vdots}{\mathsf{PRRC} \vdash q, r, \dfrac{r}{\neg p} \supset \neg\neg p}}{\neg\mathbf{L}\neg\neg p; q, r, \dfrac{r}{\neg p} \supset \neg r}(\mathsf{cD3})}{\mathbf{L}\neg p; q, r, \dfrac{r : \neg p}{\neg p} \supset \neg p}(\mathsf{cD5})}{; q, r, \dfrac{r : \neg p}{\neg p}, \dfrac{q : p}{p} \supset \neg p}(\mathsf{cD4}) \quad \dfrac{\dfrac{\vdots}{\mathsf{PRC} \vdash q, r, \dfrac{r}{\neg p} \supset \neg p}}{; q, r, \dfrac{r}{\neg p} \supset \neg p}(\mathsf{cD1})$$

Figure 5.5: Proof of "Nixon Diamond"

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \supset p}(\mathsf{id})}{\neg p, p \supset}(\supset\neg)}{\neg p \supset \neg p}(\supset\neg) \quad \supset \top}{\mathsf{PRC} \vdash \top / \neg p \supset \neg p}(\mathrm{Res2})}{\neg\mathbf{L}\neg p; \top / \neg p \supset \Delta}(\mathsf{sD2}) \quad \dfrac{\dfrac{}{p \supset}(\mathsf{aax})}{\mathsf{PRRC} \vdash \supset \neg p}(\not\supset\neg)}{\mathbf{L}\neg p; \supset \Delta}(\mathsf{sD3})$$
$$; \top : p / \neg p \supset \Delta \qquad (\mathsf{sD4})$$

Figure 5.6: Proof in theory without extension

Chapter 6

Proof Search for Residue Sequents

In this chapter we investigate two techniques for proving a residue sequent $W, R \supset A$.
We first inspect backward proof in the residue calculus from Bonatti and Olivetti [4]. For this prover we point out the redundancies and show how to use them to cut down the search tree. In a further step we develop use-check for the prover and present a calculus which implements it.
Then we discuss the second approach that relies on calculating the partial closure by calculating the minimal subsets of R whose consequents have to be added to W in order to classically prove A. A further result of the second approach is a prover that calculates the closure of a residue theory to prove a residue sequent.
We close the chapter with some experimental results to compare the different approaches and optimizations.

6.1 Backward Proof Search

In this section we investigate a prover that verifies the validity of a residue sequent by backward applying the residue rules. We start with a simple prover, show a general improvement that makes use of redundancies in the search tree and close with introducing use-check for backward proof search.

Although the calculi for default logic contain the rules of the calculi PRC and PRRC, we focus in this chapter only on the calculus PRC. We do so because PRC and PRRC are dual to each other. From an algorithmic point of view it is thus equivalent whether we fail to find a proof of a residue sequent or whether we find a refutation of it. The statements given for the

calculus PRC can thus also be formulated in a dual sense for the calculus PRRC.

6.1.1 A Simple Prover

The calculus PRC contains only two rules to conclude residues and both rules conclude sequents of the same kind. It is thus not astonishing that the rules of PRC are not invertible.

Remark 6.1 (invertibility)
(Res1) and (Res2) are not invertible.

Consider the residue sequent $p, p/q, r/s \supset q$. It is easy to see that this sequent is valid. However, neither $p, r/s \supset q$ (premise of (Res1) with principal residue p/q) nor $p, p/q \supset r$ (left premise of (Res2) with principal residue r/s) are valid.

When backward applying (Res2) then the antecedent of the left premise is replaced with the precondition of the principal residue. It is hence nearby to assume that the order in which we process the residues is of relevance. Luckily this is not the case.

Proposition 6.2 (irrelevance of processing order)
Let $W, R, A/C \supset \Delta$ be a residue sequent.
If $\vDash W, R, A/C \supset \Delta$ then $\vDash W, R \supset \Delta$, or $\vDash W, R \supset A$ and $\vDash W, C, R \supset \Delta$.

The proof of this proposition follows directly from the induction step in the proof of Theorem 5.24.

Consider the residue sequent $p_1, \dfrac{p_1}{p_2}, \dfrac{p_2}{p_3} \supset p_1 \wedge p_3$. For this sequent the proposition implies that there exist two proofs. One that concludes in its last deduction rule the residue p_1/p_2 and one that concludes p_2/p_3 (cf. Figure 6.1).

Knowing that the rules (Res1) and (Res2) are not invertible but that the order in which we process the residues is not important, we can give a simple proof search algorithm (see Algorithm 6).

6.1.2 General Improvement

During proof search we gain information about the provability of certain nodes in the search tree. The general improvement discussed in this section aims at deducing from this information the provability of other nodes. Thereby we do not use information about the sequent that we try to prove but only structural information of the search tree.

$$\dfrac{\dfrac{\dfrac{\mathbf{p_1} \supset \mathbf{p_1}}{p_1, \dfrac{\mathbf{p_2}}{\mathbf{p_3}} \supset p_1}(\text{Res1}) \qquad \dfrac{p_1, \mathbf{p_2} \supset \mathbf{p_2} \qquad \dfrac{\mathbf{p_1}, p_2, p_3 \supset \mathbf{p_1} \quad p_1, p_2, \mathbf{p_3} \supset \mathbf{p_3}}{p_1, p_2, p_3 \supset \mathbf{p_1} \wedge \mathbf{p_3}}(\supset\wedge)}{p_1, p_2, \dfrac{\mathbf{p_2}}{\mathbf{p_3}} \supset p_1 \wedge p_3}(\text{Res2})}}{p_1, \dfrac{\mathbf{p_1}}{\mathbf{p_2}}, \dfrac{p_2}{p_3} \supset p_1 \wedge p_3}(\text{Res2})$$

$$\dfrac{\dfrac{\mathbf{p_1} \supset \mathbf{p_1} \quad p_1, \mathbf{p_2} \supset \mathbf{p_2}}{p_1, \dfrac{\mathbf{p_1}}{\mathbf{p_2}} \supset p_2}(\text{Res2}) \qquad \dfrac{\dfrac{\mathbf{p_1}, p_3 \supset \mathbf{p_1} \quad p_1, \mathbf{p_3} \supset \mathbf{p_3}}{p_1, p_3 \supset \mathbf{p_1} \wedge \mathbf{p_3}}(\supset\wedge)}{p_1, \dfrac{\mathbf{p_1}}{\mathbf{p_2}}, p_3 \supset p_1 \wedge p_3}(\text{Res1})}{p_1, \dfrac{p_1}{p_2}, \dfrac{\mathbf{p_2}}{\mathbf{p_3}} \supset p_1 \wedge p_3}(\text{Res2})$$

Figure 6.1: irrelevance of processing order

Algorithm 6 Simple proof search for residue sequents

```
1: function PRCPROVABLE(W, R ⊃ Δ)
2:     if R = ∅ then
3:         success := CPCPROVABLE(W ⊃ Δ)
4:     else
5:         choose δ ∈ R and let R' := R \ {δ}
6:         success := PRCPROVABLE(W, R' ⊃ Δ)
7:         if not success then
8:             success := PRCPROVABLE(W, R' ⊃ pre(δ)) and
9:                        PRCPROVABLE(W, con(δ), R' ⊃ Δ)
10:    return success
```

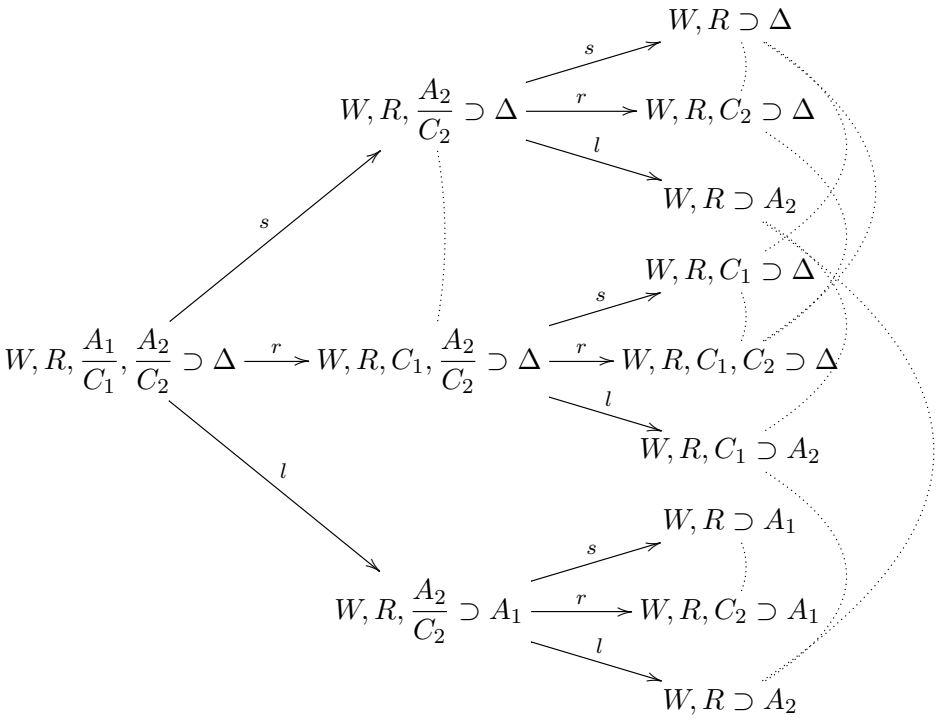


Figure 6.2: Sequents encountered in backward proof search

To discuss our approach we have depicted the sequents we encounter in proof search in Figure 6.2. Each arrow represents the processing of a premise of a residue rule. The labels s (as skip), l and r stand for the premise of (Res1), the left premise of (Res2) and the right premise of (Res2), respectively. The dotted lines connect those nodes of the same depth that have common succedents and whose antecedents are in a subset relationship.

Now the improvement we discuss is based on the monotonicity of residue theories.

Definition 6.3 (subsequent, supersequent)
Let $S_1 := \Gamma_1 \supset \Delta_1$ and $S_2 := \Gamma_2 \supset \Delta_2$ be two residue sequents. Then S_1 is a *subsequent* of S_2 and S_2 is a *supersequent* of S_1 if $\Gamma_1 \subseteq \Gamma_2$ and $\Delta_1 \subseteq \Delta_2$.

Proposition 6.4 (monotonicity)
Let $\Gamma \supset \Delta$ be a residue sequent and $\Gamma' \supset \Delta'$ be a subsequent of it.

$$\text{If } \models \Gamma' \supset \Delta' \text{ then } \models \Gamma \supset \Delta$$

Proof. The claim follows directly from Lemma 5.13.1 on page 116. □

Suppose that S_1 is a subsequent of S_2. From the previous proposition we then know that if S_1 is provable, then S_2 is provable, too. Vice versa we know that if S_2 is refutable then so is S_1. The dotted lines in Figure 6.2 thus depict situations where monotonicity can be of use to omit proving a sequent.

Two Search Strategies

The information that we gain about provable or refutable sequents during proof search can be used to omit proving other sequents. In what way this can be done depends on the proof search strategy. We illustrate this on two examples.

1. Consider the algorithm where we prioritize (Res1) to (Res2) and process the right premise of (Res2) before the left one. Then reading the sequents of Figure 6.2 top-down gives us the order in which we encounter them during proof search.

   Suppose that proving $W, R \supset \Delta$ (path (s, s)) fails, proving $W, R, C_2 \supset \Delta$ (path (s, r)) succeeds and proving $W, R \supset A_2$ (path (s, l)) fails. Then $W, R, A_2/C_2 \supset \Delta$ is not provable and we backtrack at the root of the search tree.
   If then proving $W, R, C_1 \supset \Delta$ (path (r, s)) fails we continue with the sequent $W, R, C_1, C_2 \supset \Delta$ (path (r, r)) of which the sequent at path (s, r) is a subsequent. Since this subsequent is known to be valid, we

know that $W, R, C_1, C_2 \supset \Delta$ is valid, too. Hence proving it can be omitted and we continue with sequent $W, R, C_1 \supset A_2$ (path (r, l)).

It is easy to see that with this proof search strategy only information about provable sequents are of use to omit proving other sequents.

2. Consider the algorithm where we prioritize (Res2) to (Res1) and process the right premise of (Res2) before the left one. Then reading the sequents of Figure 6.2 middle-down-up gives us the order in which we encounter them during proof search.
Suppose that proving $W, R, C_1, C_2 \supset \Delta$ (path (r, r)) fails. Since the sequent at path (r, s) is a subsequent of the one at path (r, r) it is known to be refutable, too. Hence $W, R, C_1, A_2/C_2 \supset \Delta$ (path (r)) is refutable. We thus backtrack and continue with the sequent at path (s). Now the sequent there is a subsequent of the one at path (r). Hence proving it can be omitted since it is known to be refutable, too.

It is easy to see that with this proof search strategy only information about refutable sequents are of use to omit proving other sequents.

Considering Other Search Strategies

Of course we are not bound to these two kind of proof search strategies. An obvious variation for example is to process the left premise of (Res2) before the right one.

Then there has to be no predetermined priority of what rule is firstly applied backwards. The choice which rule is to prioritize can for example be made on additional, possibly heuristic, information. Doing so could lead to an algorithm in which information about provable as well as refutable sequents might be of use to omit proving other sequents.

Furthermore the order in which the residues are processed must not be predetermined. In the two search strategies given above, we implicitly assume otherwise, but according to Algorithm 6 (line 5) this is not mandatory. Assume that when trying to prove the sequent $W, \delta, R \supset \Delta$ we chose δ as principal residue and backward apply (Res1) first. We choose an arbitrary principal residue $\delta_s \in R$ when trying to prove the premise $W, R \supset \Delta$. If later backtracking is necessary, we must backward apply (Res2) with principal residue δ. However, we may choose different principal residues $\delta_l \in R$ and $\delta_r \in R$ when proving the left and right premise. This choice may also be based on additional, possibly heuristic, information.

Implementation of General Improvement

In our implementation we use a predetermined order to process the residues. In this section we thus only describe that special case. However, adapting it to the general case is possible and easy to implement.

We have seen that during proof search we may encounter sequents that are in a subsequent relation to previously processed sequents. If we have a predetermined order in which the residues are processed then we can decide solely on the paths of two nodes in a proof search tree, whether such a relation exists between the two sequents of that nodes.

A path to a node is formally defined as a vector of the letters s, l and r, indicating the premise of the rule that was chosen at the corresponding branching depth (cf. Figure 6.2).

Definition 6.5 (PRC path)
A vector $\vec{\mathsf{p}} \in \{s, l, r\}^n$ with $n \in \mathbb{N}$ is called a PRC *path*. We denote it with $\vec{\mathsf{p}} := (\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_n)$.

We write $\text{len}(\vec{\mathsf{p}})$ to denote the *length of* $\vec{\mathsf{p}}$ and $\vec{\mathsf{p}}[i]$ to denote the i'th element of $\vec{\mathsf{p}}$, where $1 \leq i \leq \text{len}(\mathsf{p})$.

For two paths $\vec{\mathsf{p}} := (\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_n)$ and $\vec{\mathsf{q}} := (\mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_m)$ we write $\vec{\mathsf{p}} \circ \vec{\mathsf{q}}$ to denote their concatenation $(\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_n, \mathsf{q}_1, \mathsf{q}_2, \ldots, \mathsf{q}_m)$.

In our implementation we use a vector $\vec{R}$ of residues that we process from its first to its last element. This way we have a given order in which the residues are processed. $\vec{R}$ corresponds to a set R of residues, i.e. $\text{len}(\vec{R}) = |R|$ and $R = \left\{\vec{R}[i] : 1 \leq i \leq \text{len}(\vec{R})\right\}$. Our algorithms thus operate over sequents of the form $W, \vec{R} \supset \Delta$. In the following we use $\vec{R}$ in a sequent to indicate a predetermined order in which the residues are processed.

During proof search of a residue sequent S the sequent at a given path $\vec{\mathsf{p}}$ of the search tree can be computed from S and $\vec{\mathsf{p}}$. This is done according to three cases.

1. When processing the premise of (Res1) (an s in $\vec{\mathsf{p}}$), we remove the principal residue.
2. When processing the left premise of (Res2) (an l in $\vec{\mathsf{p}}$), we remove the principal residue and set its prerequisite as the new succedent.
3. When processing the right premise of (Res2) (an r in $\vec{\mathsf{p}}$), we replace the principal residue with its consequent.

The following definition is used to compute the sequent at a given path from the sequent that is to prove.

Definition 6.6 (ant($\vec{\mathsf{p}}$), suc($\vec{\mathsf{p}}$))
For a PRC path $\vec{\mathsf{p}} := (\mathsf{p}_1, \ldots, \mathsf{p}_n)$ we define:

$$\begin{aligned} \text{ant}(\vec{\mathsf{p}}) &:= \{i : 1 \leq i \leq \text{len}(\vec{\mathsf{p}}) \text{ and } \mathsf{p}[i] = r\} \\ \text{suc}(\vec{\mathsf{p}}) &:= \max(\{i : 1 \leq i \leq \text{len}(\vec{\mathsf{p}}) \text{ and } \mathsf{p}[i] = l\} \cup \{0\}) \end{aligned}$$

The function ant($\vec{\mathsf{p}}$) returns for a path $\vec{\mathsf{p}}$ the indices of those elements in $\vec{R}$, whose consequent replace the corresponding residue in S.

The function suc($\vec{\mathsf{p}}$) serves to calculate the succedent. If it is equal to 0 then it indicates that the succedent is the same as in S. Otherwise it gives us the index of that element in $\vec{R}$, whose prerequisite forms the succedent.

We use ant($\vec{\mathsf{p}}$) and suc($\vec{\mathsf{p}}$) to calculate from a path $\vec{\mathsf{p}}$ in the search tree of the proof of $W, \vec{R} \supset \Delta$ the sequent $\Gamma_{\vec{\mathsf{p}}}(W, \vec{R}) \supset \Delta_{\vec{\mathsf{p}}}(\Delta)$ encountered at $\vec{\mathsf{p}}$.

Definition 6.7 ($\Gamma_{\vec{\mathsf{p}}}(W, \vec{R})$, $\Delta_{\vec{\mathsf{p}}}(\Delta, \vec{R})$)
Let W and Δ be sets of formulas, $\vec{R}$ a vector of residues and $\vec{\mathsf{p}}$ a PRC path with $\text{len}(\vec{\mathsf{p}}) \leq \text{len}(\vec{R})$. Then we define the multisets $\Gamma_{\vec{\mathsf{p}}}(W, \vec{R}) \subseteq \mathcal{L}^{res}$ and $\Delta_{\vec{\mathsf{p}}}(\Delta, \vec{R}) \subseteq \mathcal{L}$ as follows.

$$\Gamma_{\vec{\mathsf{p}}}(W, \vec{R}) := W \cup \left\{\text{con}(\vec{R}[i]) : i \in \text{ant}(\vec{\mathsf{p}})\right\} \cup \left\{\vec{R}[i] : \text{len}(\vec{\mathsf{p}}) < i \leq \text{len}(\vec{R})\right\}$$

$$\Delta_{\vec{\mathsf{p}}}(\Delta, \vec{R}) := \begin{cases} \Delta & \text{if suc}(\vec{\mathsf{p}}) = 0 \\ \text{pre}(\vec{R}[\text{suc}(\vec{\mathsf{p}})]) & \text{otherwise} \end{cases}$$

This definition leads to a relation on paths which reflects the subsequent relation of the sequents in the search tree.

Definition 6.8 ($\vec{\mathsf{p}} \leq \vec{\mathsf{q}}$)
For two PRC paths $\vec{\mathsf{p}}$ and $\vec{\mathsf{q}}$ we define the relation $\leq$ as follows:

$$\vec{\mathsf{p}} \leq \vec{\mathsf{q}} \quad \text{iff} \quad \text{ant}(\vec{\mathsf{p}}) \subseteq \text{ant}(\vec{\mathsf{q}}), \text{suc}(\vec{\mathsf{p}}) = \text{suc}(\vec{\mathsf{q}}) \text{ and } \text{len}(\vec{\mathsf{p}}) \geq \text{len}(\vec{\mathsf{q}}).$$

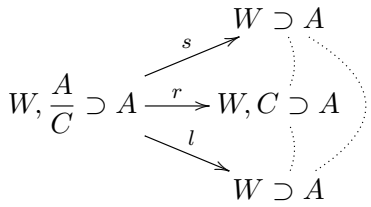
Now if for two paths $\vec{\mathsf{p}}$ and $\vec{\mathsf{q}}$ we have $\vec{\mathsf{p}} \leq \vec{\mathsf{q}}$ then we know that the sequent at $\vec{\mathsf{p}}$ is a subsequent of the one at $\vec{\mathsf{q}}$.

Theorem 6.9 ($\vec{\mathsf{p}} \leq \vec{\mathsf{q}}$ **implies a subsequent relation**)
Let W and Δ be sets of formulas, $\vec{R}$ be a vector of residues and $\vec{\mathsf{p}}$ and $\vec{\mathsf{q}}$ be PRC paths with $\text{len}(\vec{\mathsf{p}}) \leq \text{len}(\vec{R})$ and $\text{len}(\vec{\mathsf{q}}) \leq \text{len}(\vec{R})$.

If $\vec{\mathsf{p}} \leq \vec{\mathsf{q}}$ then $\Gamma_{\vec{\mathsf{p}}}(W, \vec{R}) \supset \Delta_{\vec{\mathsf{p}}}(\Delta, \vec{R})$ is a subsequent of $\Gamma_{\vec{\mathsf{q}}}(W, \vec{R}) \supset \Delta_{\vec{\mathsf{q}}}(\Delta, \vec{R})$.

Proof. Let $\vec{\mathsf{p}} \leq \vec{\mathsf{q}}$ (1). Then we know that $\text{ant}(\vec{\mathsf{p}}) \subset \text{ant}(\vec{\mathsf{q}})$ and $\text{len}(\vec{\mathsf{p}}) \geq \text{len}(\vec{\mathsf{q}})$, hence $\Gamma_{\vec{\mathsf{p}}}(W, \vec{R}) \subseteq \Gamma_{\vec{\mathsf{q}}}(W, \vec{R})$. From (1) we also know that $\text{suc}(\vec{\mathsf{p}}) = \text{suc}(\vec{\mathsf{q}})$, hence $\Delta_{\vec{\mathsf{p}}}(W, \vec{R}) = \Delta_{\vec{\mathsf{q}}}(W, \vec{R})$. □

The inverse of this theorem is not valid. The following situation reflects such a case in which the sequents at the paths (s) and (l) are in a subsequent relation but we neither have $(s) \leq (l)$ nor $(l) \leq (s)$.



This example raises the question why we use only structural information for the general improvement. Using the sequents themselves instead of their paths could indeed lead to more situations where proving a sequent can be omitted. However, there are some points in favor of using paths. First, to compare sequents efficiently, we would need to sort them, e.g. with the help of sorted containers. Now inserting an element into a sorted container is more time-consuming than updating a path. Second, comparing two sequents means comparing their formulas, which is also more time-consuming than comparing two paths. Third, instead of storing the paths of the nodes that are known to be provable or refutable we would have to perform the more complex task to store the corresponding sequents. All in all these were good arguments to use the solution presented above.

Algorithm 7 Proof search with general improvement for residue sequents

```
1: function prcProvableG(W, R⃗ ⊃ Δ, p⃗, pos)
2:     if q⃗ ≤ p⃗ for some q⃗ ∈ pos then
3:         return true
4:     else if len(R⃗) = len(p⃗) then
5:         success := cpcProvable(Γ_p⃗(W, R⃗) ⊃ Δ_p⃗(Δ, R⃗))
6:     else
7:         success := prcProvableG(W, R⃗ ⊃ Δ, p⃗ ∘ (s), pos)
8:         if not success then
9:             success := prcProvableG(W, R⃗ ⊃ Δ, p⃗ ∘ (r), pos) and
10:                        prcProvableG(W, R⃗ ⊃ Δ, p⃗ ∘ (l), pos)
11:    if success then
12:        pos := pos ∪ {p⃗}
13:    return success
```

The pseudocode of the first search strategy above is given in Algorithm 7. As already mentioned, only information about provable sequents are of importance for this search strategy. The proving function therefore takes as arguments the sequent $W, \vec{R} \supset \Delta$ that is to prove, the current position

$\vec{\mathsf{p}}$ in the search tree and a set pos of paths whose sequents are know to be provable. Path $\vec{\mathsf{p}}$ is passed by value and pos is passed by reference.

On line 2 we check whether the sequent at the given position is know to be provable. If it is, we immediately return with success. Otherwise we continue with the proof search process. If there are no residues left in the sequent (line 4) we call the CPC prover (line 5). Otherwise we backward apply the non-branching rule (line 7) and if this does not succeed the branching rule (lines 9+10). If the sequent at the current path $\vec{\mathsf{p}}$ is provable, we add $\vec{\mathsf{p}}$ to the set pos of provable paths (lines 11–12).

6.1.3 Use-Check

In this section we present use-check for the calculus PRC. As in use-check for CPC we take advantage of the fact that the fragment of default logic restricted to residues is monotonic (see Proposition 6.4 on page 131). Because of that, use-check for PRC turns out to be more or less equivalent to use-check for CPC.

We start with an introductory example to discuss some aspects of our approach. Then we present a modified calculus with inbuilt use-check which is based on labeled residue sequents. We close this section by pointing out how to combine use-check and the general improvement.

Introductory Example

Use-check for the residue sequent calculus concerns rule (Res2) and is similar to use-check for the classical calculus. Suppose that in our proof search there is a backward rule application of (Res2).

$$\frac{\begin{array}{cc}\vdots & \vdots \\ W, R \supset A & W, R, C \supset \Delta\end{array}}{W, R, A/C \supset \Delta}\text{(Res2)}$$

Then there are two cases where use-check can successfully be applied.

1. The proof search procedure is successful in proving the left premise. If analyzing this proof reveals that already $W, R \supset$ is valid, then by monotonicity we know that the conclusion $W, R, A/C \supset \Delta$ is valid. Hence proving the right premise can be omitted.
2. The proof search procedure is successful in proving the right premise. If analyzing this proof reveals that already $W, R \supset \Delta$ is valid, then by monotonicity we know that the conclusion $W, R, A/C \supset \Delta$ is valid. Hence proving the left premise can be omitted.

If use-check is successful in the first case, i.e. if $W, R \supset$ is known to be valid, then the validity of the non-proven premise $W, R, C \supset \Delta$ follows by monotonicity.
We have a different situation in the second case. If use-check is successful in the right premise, i.e. if $W, R \supset \Delta$ is known to be valid, then we can not conclude the validity of the left premise $W, R \supset A$ from it. Nevertheless we know that the conclusion is valid. This case reflects the situation where (Res1) instead of (Res2) should be used.

To detect whether use-check can be applied, we follow the same approach as in classical logic. Regarding the first case this means that use-check is applicable if no formula originating in A is used as principal formula in an axiom.

Consider the following proof fragment.

$$\dfrac{\dfrac{\vdots}{\Gamma, p, p \to q \supset A} \quad \dfrac{\dfrac{}{\Gamma, \mathbf{p}, s \supset \mathbf{p}, q}(\mathsf{id}) \quad \dfrac{}{\Gamma, p, \mathbf{q}, s \supset \mathbf{q}}(\mathsf{id})}{\Gamma, p, \mathbf{p} \to \mathbf{q}, s \supset q}(\to\supset)}{\Gamma, p, p \to q, \mathbf{A/s} \supset q}(\text{Res2})$$

The variable s is not used in the axioms of the proof of the right premise. In this simple example we can just remove s from all of its sequents and obtain like this a proof of $\Gamma, p, p \to q \supset q$. Using (Res1) we then obtain a proof of $\Gamma, p, p \to q, A/s \supset q$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \mathbf{p} \supset \mathbf{p}, q}(\mathsf{id}) \quad \dfrac{}{\Gamma, p, \mathbf{q} \supset \mathbf{q}}(\mathsf{id})}{\Gamma, p, \mathbf{p} \to \mathbf{q} \supset q}(\to\supset)}{\Gamma, p, p \to q, \mathbf{A/s} \supset q}(\text{Res1})$$

A More Efficient Residue Calculus

Because of multiple formula occurrences it is not always clear from which formula or default rule a principal formula in an axiom originates. This problem is already known from classical logic (see page 29). To avoid that ambiguity we use the same approach as in classical logic. That is we put labels on residues and formulas and store the information about relevant formulas and residues in a set of labels. Like this, use-check for the residue calculus can be implemented analogous to classical logic (see page 31).

Since the mechanism of use-check for the residue calculus is the same as for the classical propositional calculus, we do without further explanations and proceed with the necessary definitions for a residue calculus with inbuilt use-check.

Definition 6.10 (labeled default rule)
A *labeled default rule* is a pair $\langle \delta, \mathrm{n} \rangle$ consisting of a default rule δ and a natural number $\mathrm{n} \in \mathbb{N}$, called the *label*. It is denoted by $\overset{\mathrm{n}}{\delta}$.

A labeled default rule with an empty justification is called a *labeled residue*.

We write $\mathcal{L}_{\mathbb{N}}^{res}$ to denote the set of labeled formulas and labeled residues and $\mathcal{L}_{\mathbb{N}}^{def}$ to denote the set of labeled formulas and labeled defaults.

To write down a labeled default with its prerequisite, justifications and consequent we use the same notation as for default rules but with a label on top of the colon separating the prerequisite from the justifications.

$$\frac{A \overset{1}{:} B_1, B_2}{C} \quad A \overset{1}{:} B_1, B_2/C \quad \frac{\overset{2}{:} B_1, B_2}{C} \quad \overset{2}{:} B_1, B_2/C \quad \frac{A \overset{3}{:}}{C} \quad A \overset{3}{:}/C$$

To denote sets or multisets of labeled defaults we write $\overline{R}$ and $\overline{D}$. For sets or multisets of labeled formulas and defaults we use the symbol $\overline{\Gamma}$.

Definition 6.11 (labels$(\overline{\Gamma})$**,** $\overline{\Gamma}{\downarrow_{\mathcal{D}}}$**)**
Let $\overline{\Gamma} \subseteq \mathcal{L}_{\mathbb{N}}^{def}$ be a set of multiset of labeled formulas and defaults. We define the set labels$(\overline{\Gamma})$ of *labels of* $\overline{\Gamma}$ as:

$$\mathrm{labels}(\overline{\Gamma}) := \mathrm{labels}(\overline{\Gamma} \cap \mathcal{L}_{\mathbb{N}}) \cup \left\{ \mathrm{n} : \overset{\mathrm{n}}{\delta} \in \overline{\Gamma} \setminus \mathcal{L}_{\mathbb{N}} \text{ for some } \mathrm{n} \in \mathbb{N} \right\}.$$

Furthermore we define the set $\overline{\Gamma}{\downarrow_{\mathcal{D}}} \subseteq \mathcal{L}^{def}$ as

$$\overline{\Gamma}{\downarrow_{\mathcal{D}}} := (\overline{\Gamma} \cap \mathcal{L}_{\mathbb{N}}){\downarrow_{\mathcal{D}}} \cup \left\{ \delta : \text{there exists } \mathrm{n} \in \mathcal{D} \text{ such that } \overset{\mathrm{n}}{\delta} \in \overline{\Gamma} \setminus \mathcal{L}_{\mathbb{N}} \right\}.$$

Definition 6.12 (labeled default sequent)
A *labeled default sequent* is a triple $\langle \mathcal{D}, \overline{\Gamma}, \overline{\Delta} \rangle$ consisting of a finite multiset $\overline{\Gamma} \subset \mathcal{L}_{\mathbb{N}}^{def}$, a finite multiset $\overline{\Delta} \subset \mathcal{L}_{\mathbb{N}}$ and a use-set $\mathcal{D} \subseteq \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$.

We write $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ to denote a labeled default sequent.

$\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is called a *labeled residue sequent* if $\overline{\Gamma}$ is a finite multiset of $\mathcal{L}_{\mathbb{N}}^{res}$.

Definition 6.13 (valid labeled default sequent)
A labeled default sequent $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is defined to be valid if $\overline{\Gamma}{\downarrow_{\mathcal{D}}} \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ is valid. We denote this as usual with $\vDash \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$.

Definition 6.14 (the calculus PRC2)
We define the residue calculus PRC2 to have the deduction rules of CPC2 and those given in Figure 6.3.

The motivation of the four residue rules of PRC2 is as follows.

$$\frac{\mathcal{D};\overline{\Gamma}\supset\overline{\Delta}}{\mathcal{D};\overline{\Gamma},\dfrac{A\overset{\mathrm{m}}{\vdots}}{C}\supset\overline{\Delta}}(\text{Res1}) \qquad \frac{\mathcal{D};\overline{\Gamma}\supset\overset{\mathrm{n}}{A}}{\mathcal{D};\overline{\Gamma},\dfrac{A\overset{\mathrm{m}}{\vdots}}{C}\supset\overline{\Delta}}(\text{Res2p})^{a} \qquad \frac{\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{C}\supset\overline{\Delta}}{\mathcal{D};\overline{\Gamma},\dfrac{A\overset{\mathrm{m}}{\vdots}}{C}\supset\overline{\Delta}}(\text{Res2c})^{a}$$

$$\frac{\mathcal{D}_1,\mathrm{n};\overline{\Gamma}\supset\overset{\mathrm{n}}{A}\quad\mathcal{D}_2,\mathrm{n};\overline{\Gamma},\overset{\mathrm{n}}{C}\supset\overline{\Delta}}{\mathcal{D}_1,\mathcal{D}_2,\mathrm{m},\mathrm{n}^{*};\overline{\Gamma},\dfrac{A\overset{\mathrm{m}}{\vdots}}{C}\supset\overline{\Delta}}(\text{Res2})^{b}$$

[a] $\mathrm{n}\notin\mathcal{D}$
[b] $\mathrm{n}\notin\mathcal{D}_1\cup\mathcal{D}_2$
[*] if $\mathrm{n}\in\text{labels}(\overline{\Gamma}\cup\overline{\Delta})$, otherwise without n

Figure 6.3: Residue rules of PRC2

- (Res1): This rule is equivalent to the rule of the same name of PRC.
- (Res2p) and (Res2c): these rules reflect successful use-checks in the left and right premise of (Res2), respectively. The condition to apply these rules is therefore that the label n of the active formula in the premise must not occur in the use-set $\mathcal{D}$.
- (Res2): This rule reflects the case where use-check is not successful. In both its premises the label n of the active formula in the premise occurs in the corresponding use-set. To form the use-set of the conclusion we merge the two use-sets of the premises and add the label of the principal formula. The label n of the active formulas in the premises is only added to the use-set of the conclusion if it occurs on other formulas in the premises.

Remark 6.15 (CPC2 rules in PRC2)
The rules of CPC2 in PRC2 may only be applied on pure CPC2-sequents, i.e. on sequents that do not have a labeled residue in its antecedent.

Theorem 6.16 (soundness of PRC2)
If a labeled residue sequent $\mathcal{D};\overline{\Gamma}\supset\overline{\Delta}$ is deducible in PRC2 then is it valid.

Proof. Let $\mathcal{D};\overline{\Gamma}\supset\overline{\Delta}$ be a labeled residue sequent deduced in PRC2, $\overline{W}$ the labeled formulas of $\overline{\Gamma}$ and $\overline{R}$ the labeled residues of $\overline{\Gamma}$.
We show our claim by induction on the number r of residues in $\overline{\Gamma}$.

r = 0**:** Then the validity of $\mathcal{D};\overline{\Gamma}\supset\overline{\Delta}$ follows from the soundness of CPC2.

r = k + 1**:** Then $\overline{\Gamma}$ is of the form $\overline{\Gamma}',A\overset{\mathrm{m}}{\vdots}/C$. We distinguish on the last rule in the proof.

- $$\frac{\mathcal{D};\overline{\Gamma}' \supset \overline{\Delta}}{\mathcal{D};\overline{\Gamma}', A \overset{\mathrm{m}}{:} /C \supset \overline{\Delta}}(\text{Res1})$$

  Then $\mathcal{D};\overline{\Gamma}' \supset \overline{\Delta}$ is valid by induction hypothesis and we thus know by definition that $\overline{\Gamma}'{\downarrow_{\mathcal{D}}} \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ is valid. From Proposition 6.4 we then know that $\overline{\Gamma}'{\downarrow_{\mathcal{D}}}, A/C \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ is valid. Hence, independent of whether m is an element of $\mathcal{D}$ or not, we know that the conclusion $\mathcal{D};\overline{\Gamma}', A \overset{\mathrm{m}}{:} /C \supset \overline{\Delta}$ is valid.

- $$\frac{\mathcal{D};\overline{\Gamma}' \supset \overset{\mathrm{n}}{A}}{\mathcal{D};\overline{\Gamma}', \dfrac{A \overset{\mathrm{m}}{:}}{C} \supset \overline{\Delta}}(\text{Res2p}), \text{ with } \mathrm{n} \notin \mathcal{D}$$

  Since $\mathrm{n} \notin \mathcal{D}$ we know $\vDash \overline{\Gamma}'{\downarrow_{\mathcal{D}}} \supset$. With Proposition 6.4 we obtain $\vDash \overline{\Gamma}'{\downarrow_{\mathcal{D}}} \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ and $\vDash \overline{\Gamma}'{\downarrow_{\mathcal{D}}}, A/C \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$. Hence, independent of whether m is an element of $\mathcal{D}$ or not, we know that the conclusion $\mathcal{D};\overline{\Gamma}', A \overset{\mathrm{m}}{:} /C \supset \overline{\Delta}$ is valid.

- $$\frac{\mathcal{D};\overline{\Gamma}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}}{\mathcal{D};\overline{\Gamma}', \dfrac{A \overset{\mathrm{m}}{:}}{C} \supset \overline{\Delta}}(\text{Res2c}), \text{ with } \mathrm{n} \notin \mathcal{D}$$

  Analogous to the previous case.

- $$\frac{\mathcal{D}_1, \mathrm{n};\overline{\Gamma}' \supset \overset{\mathrm{n}}{A} \quad \mathcal{D}_2, \mathrm{n};\overline{\Gamma}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}}{\mathcal{D}_1, \mathcal{D}_2, \mathrm{m}, \mathrm{n}^*;\overline{\Gamma}', A \overset{\mathrm{m}}{:} /C \supset \overline{\Delta}}(\text{Res2}), \text{ cf. Figure 6.3 for requirements}$$

  Then $\mathcal{D}_1, \mathrm{n};\overline{\Gamma}' \supset \overset{\mathrm{n}}{A}$ and $\mathcal{D}_2, \mathrm{n};\overline{\Gamma}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}$ are valid by induction hypothesis. By definition we thus know

  $$\vDash \overline{\Gamma}'{\downarrow_{\mathcal{D}_1 \cup \{\mathrm{n}\}}} \supset A \ (1) \quad \text{and} \quad \vDash \overline{\Gamma}'{\downarrow_{\mathcal{D}_2 \cup \{\mathrm{n}\}}}, C \supset \overline{\Delta}{\downarrow_{\mathcal{D}_2 \cup \{\mathrm{n}\}}} \ (2).$$

  $$\text{Let } \mathcal{D} := \begin{cases} \mathcal{D}_1 \cup \mathcal{D}_2 \cup \{\mathrm{m}, \mathrm{n}\} & \text{if } \mathrm{n} \in \text{labels}(\overline{\Gamma}' \cup \overline{\Delta}), \\ \mathcal{D}_1 \cup \mathcal{D}_2 \cup \{\mathrm{m}\} & \text{if } \mathrm{n} \notin \text{labels}(\overline{\Gamma}' \cup \overline{\Delta}). \end{cases}$$

  From (1) and (2) we obtain with Proposition 6.4 that $\overline{\Gamma}'{\downarrow_{\mathcal{D}}} \supset A$ and $\overline{\Gamma}'{\downarrow_{\mathcal{D}}}, C \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ are valid, hence $\overline{\Gamma}'{\downarrow_{\mathcal{D}}}, A/C \supset \overline{\Delta}{\downarrow_{\mathcal{D}}}$ is valid and therefore $\mathcal{D};\overline{\Gamma}', A \overset{\mathrm{m}}{:} /C \supset \overline{\Delta}$ is valid, too.

□

Theorem 6.17 (completeness of PRC2)
Let $\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}$ be a labeled residue sequent.

If $\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}$ is valid, then it is deducible in PRC2.

Proof. Let $\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}$ be a valid labeled residue sequent, $\overline{W}$ the labeled formulas of $\overline{\Gamma}$ and $\overline{R}$ the labeled residues of $\overline{\Gamma}$.
We show our claim by induction on $|\overline{R}|$.

$|\overline{R}| = 0$**:** Then we know from the completeness of CPC2 that $\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}$ is deducible.

$|\overline{R}| = \mathrm{r}+1$**:** Let $A \overset{\mathrm{m}}{\vdots} /C \in \overline{R}$ and $\overline{R}' := \overline{R} \setminus \left\{A \overset{\mathrm{m}}{\vdots} /C\right\}$. Furthermore let $W := \overline{W}{\downarrow_{\mathcal{D}}}$, $R := \overline{R}{\downarrow_{\mathcal{D}}}$, and $R' := \overline{R}'{\downarrow_{\mathcal{D}}}$.

From the validity of $\mathcal{D};\overline{W},\overline{R} \supset \overline{\Delta}$ we know by definition that $W, R \supset \Delta$ is valid.

We distinguish two cases

- $\mathrm{m} \notin \mathcal{D}$: Then $R = R'$, i.e. $W, R \supset \Delta$ and $W, R' \supset \Delta$ are equal. Hence $\mathcal{D};\overline{W},\overline{R}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}$ is valid (especially for $\mathrm{n} \notin \mathcal{D}$) and by induction hypothesis deducible in PRC2. With (Res2c) we can deduce $\mathcal{D};\overline{W},\overline{R} \supset \overline{\Delta}$.
- $\mathrm{m} \in \mathcal{D}$: Then $W, R', A/C \supset \Delta$ is valid. This is equivalent to $\bigvee \Delta \in \mathrm{Cl}(W, R' \cup \{A/C\})$ $(*)$.

  We distinguish two cases:

  - $A \notin \mathrm{Cl}(W, R')$: Then by Lemma 5.14 $\mathrm{Cl}(W, R' \cup \{A/C\}) = \mathrm{Cl}(W, R')$. Hence $\vDash W, R' \supset \Delta$ and thus $\mathcal{D};\overline{W},\overline{R}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}$ is valid, especially for $\mathrm{n} \notin \mathcal{D}$. The latter is deducible in PRC2 by induction hypothesis. With (Res2p) we can deduce $\mathcal{D};\overline{W},\overline{R}', A \overset{\mathrm{m}}{\vdots} /C \supset \overline{\Delta}$.
  - $A \in \mathrm{Cl}(W, R')$: Then $\vDash W, R' \supset A\,(1)$. Hence, by Lemma 5.14, we know $\mathrm{Cl}(W, R' \cup \{A/C\}) = \mathrm{Cl}(W \cup \{C\}, R')$ and therefore from $(*) \vDash W, R', C \supset \Delta\,(2)$.

    Let $\mathcal{D}' := \mathcal{D} \setminus \{\mathrm{n}\}$. From (1) and (2) we get $\vDash \mathcal{D}', \mathrm{n}; \overline{W}, \overline{R}' \supset \overset{\mathrm{n}}{A}$ and $\vDash \mathcal{D}', \mathrm{n}; \overline{W}, \overline{R}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}$. By induction hypothesis these two labeled residue sequents are deducible in PRC2. With (Res2) we can deduce $\mathcal{D};\overline{W},\overline{R}', A \overset{\mathrm{m}}{\vdots} /C \supset \overline{\Delta}$

□

Remark 6.18 (superfluous rules)
In the proof of the completeness of PRC2 we do not use rule (Res1). This rules is in fact superfluous and only present in PRC2 because one of the proof search strategies we investigate is to skip residues before taking them into consideration and there we need rule (Res1) because backward applying it represents skipping a residue.

A Proof Search Algorithm With Use-Check

From the calculus PRC2 we can easily extract a proof search algorithm that does use-check (cf. Algorithm 8).

Algorithm 8 Proof search with use-check for residue sequents

```
1:  function PRC2PROVABLE(𝒟; W̄, R̄ ⊃ Δ̄)
2:      if R̄ = ∅ then
3:          success := CPC2PROVABLE(𝒟; W̄ ⊃ Δ̄)
4:      else
5:          choose A ⋮^m/C ∈ R̄ and let R̄′ := R̄ \ {A ⋮^m/C}
6:          success := PRC2PROVABLE(𝒟; W̄, R̄′ ⊃ Δ̄)
7:          if not success then
8:              choose a fresh label n ∈ ℕ
9:              success := PRC2PROVABLE(𝒟1; W̄, R̄′ ⊃ A^n)
10:             if n ∉ 𝒟1 then
11:                 𝒟 := 𝒟1
12:             else
13:                 success := PRC2PROVABLE(𝒟2; W̄, R̄′, C^n ⊃ Δ̄)
14:                 if n ∉ 𝒟2 then
15:                     𝒟 := 𝒟2
16:                 else
17:                     𝒟 := (𝒟1 ∪ 𝒟2 ∪ {m}) \ {n}
18:     return success
```

The recursive prover function takes as argument the labeled residue sequent that is to prove. The use-set $\mathcal{D}$ is thereby computed when returning from the recursive calls.

The layout of the algorithm is as follows.

The first block (lines 2–3) treats the base case of the recursion. If there are no residues in the sequent to prove, we call the CPC2 prover. If it succeeds we obtain a use-set $\mathcal{D}$ with the labels of those formulas that are needed for the proof, otherwise $\mathcal{D}$ is empty.

The second block (lines 4–17) treats the recursion step, i.e. proof search on the residue rules. We choose an arbitrary principal residue (line 5) and try to apply rule (Res1) backwards with a recursive function call (line 6). If this fails (line 7) we backtrack and try to apply rule (Res2) backwards. There we choose a label $\mathrm{n} \in \mathbb{N}$ for the active formulas in the premises (line 8). For optimal use-check this should be a number that is not yet used as a label in the given sequent. We start by trying to prove the left premise (line 9). Now depending on the use-set $\mathcal{D}_1$ obtained from the recursive function call,

we distinguish two cases.

1. If n $\notin \mathcal{D}_1$ (line 10) then either proving the left premise failed ($\mathcal{D}_1 = \emptyset$) or proving it succeeded and use-check is successful (rule (Res2p)). In both cases we set $\mathcal{D}$ to be $\mathcal{D}_1$ and return from the function.

2. If n $\in \mathcal{D}_1$ (line 12) then the left premise was provable and use-check is not successful. We must continue with proving the right premise (line 13). Again we distinguish two cases depending on the use-set $\mathcal{D}_2$ obtained from the recursive function call.

   (a) If n $\notin \mathcal{D}_2$ (line 14) then either proving the right premise failed ($\mathcal{D}_2 = \emptyset$) or proving it succeeded and use-check is successful (rule (Res2c)). In both cases we set $\mathcal{D}$ to be $\mathcal{D}_2$ and return from the function.

   (b) If n $\in \mathcal{D}_2$ (line 16) then the right premise was provable and use-check is not successful (rule (Res2)). We calculate the use-set $\mathcal{D}$ accordingly and return from the function.

Algorithmic Strategies

In the previously sketched algorithms we prefer rule (Res1) to rule (Res2). We do not know of an optimal strategy telling us when to prioritize the one rule above the other. However, with the use-check mechanism we tend to prioritize (Res2) for the following reasons.

Suppose that we want to prove $\Gamma, A/C \supset \Delta$ and choose A/C as principal residue. Dependent on the validity of the sequent, we have different optimal or advantageous proof strategies.

- $\Gamma, A/C \supset \Delta$ is valid.

  Then we have two cases.

  1. $\Gamma \supset \Delta$ is valid: Then applying (Res1) first is optimal.

     Together with use-check, applying (Res2) first may also be a good choice. In this case the proof of the right premise $\Gamma, C \supset \Delta$ is known to succeed and since $\Gamma \supset \Delta$ is valid, it is possible that use-check is successful here. If use-check succeeds, we end up with about the same proving effort as in the optimal case. If use-check fails then we need to prove the left premise and may need to backtrack.

  2. $\Gamma \supset \Delta$ is not valid: Then applying (Res2) first is optimal.

In this case we know that the two premises $\Gamma \supset A$ and $\Gamma, C \supset \Delta$ are valid and that use-check will fail in both proofs. Whether we first try to prove the left or the right premise does not matter.

- $\Gamma, A/C \supset \Delta$ is not valid.

  Then we know that $\Gamma \supset \Delta$ is not valid, and $\Gamma \supset A$ or $\Gamma, C \supset \Delta$ are not valid.

  If we start with the right premise $\Gamma, C \supset \Delta$ of (Res2) and proving it fails, then we know that proving $\Gamma \supset \Delta$ will fail, too. We can thus omit backtracking.

  If we start with the left premise $\Gamma \supset A$ of (Res2) and proving it fails, then proving the right premise can be omitted. However, we have to backtrack and try rule (Res1).

  If we start with rule (Res1) then this will fail and we will have to backtrack and try one of the two premises of (Res2). This either fails or succeeds. In the former case we can return with failure, in the latter case we will have to try to prove the other premise, too.

  In this case it is hence advantageous to try proving the right premise of rule (Res2) first.

Combining General Improvement And Use-Check

The general improvement and use-check may be combined. When omitting a proof of a sequent due to general improvement we must take care that we provide a use-set for the sequent whose proof has been omitted.
We illustrate this on a proof of $A_1, C_1 \to A_2, A_1/C_1, A_2/C_2 \supset C_2$ and consider a top down search strategy according to Figure 6.2 on page 130. Some of the sequents we prove during proof search are given in figure 6.4.

Proving $\overset{1}{A_1}, C_1 \overset{2}{\to} A_2, A_2 \overset{4}{\vdots} /C_2 \supset \overset{5}{C_2}$ (path (s)) fails because the sequents at path $(s.s)$ and $(s.l)$ are invalid. However, proving the sequent at path $(s.r)$ succeeds with use-set $\mathcal{D}_{sr} = \{4, 5\}$. We have to backtrack.
First proving $\overset{1}{A_1}, C_1 \overset{2}{\to} A_2, \overset{3}{C_1} \supset \overset{5}{C_2}$ (path (r, s)) fails. Then the sequent at path (r, r) is known to be valid because it is a supersequent of the one at path (s, r). We hence do not prove it and thus do not obtain a use-set for it. Proving the sequent at path (r, l) succeeds with use-seq $\mathcal{D}_{rl} = \{1, 4\}$. Hence the sequent at path (r) is valid. In order to compute a use-set for it we need a use-set of the sequent at path (r, r). Since the sequent there is a supersequent of the one at path (s, r) it is easy to see, that $\mathcal{D}_{sr}$ is an appropriate use-set for it. We can hence not only inherit the validity but also the use-set of the sequent at position (s, r).

$$
\begin{array}{lll}
\overset{1}{A}_1, C_1 \overset{2}{\to} A_2, A_1 \overset{3}{:}/C_1, A_2 \overset{4}{:}/C_2 \supset \overset{5}{C}_2 & & \\
(s)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2, A_2 \overset{4}{:}/C_2 \supset \overset{5}{C}_2 & & \\
(s,s)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2 \supset \overset{5}{C}_2 & \bot & \\
(s,r)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2, \overset{4}{C}_2 \supset \overset{5}{C}_2 & \top & \mathcal{D}_{sr} = \{4,5\} \\
(s,l)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2, \supset \overset{4}{A}_2 & \bot & \\
(r)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2, \overset{3}{C}_1, A_2 \overset{4}{:}/C_2 \supset \overset{5}{C}_2 & & \\
(r,s)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2, \overset{3}{C}_1 \supset \overset{5}{C}_2 & \bot & \\
(r,r)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2, \overset{3}{C}_1, \overset{4}{C}_2 \supset \overset{5}{C}_2 & (\top) & \mathcal{D}_{rr} = \mathcal{D}_{sr} = \{4,5\} \\
(r,l)\ \overset{1}{A}_1, C_1 \overset{2}{\to} A_2, \overset{3}{C}_1 \supset \overset{4}{A}_2 & \top & \mathcal{D}_{rl} = \{1,4\} \\
\vdots & &
\end{array}
$$

Figure 6.4: Combining general improvement and use-check

The following proposition expresses the situation formally.

Proposition 6.19 (monotonicity of labeled residue sequent)
Let $\mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ and $\mathcal{D}'; \overline{\Gamma}' \supset \overline{\Delta}'$ be labeled residue sequents.

If $\vDash \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$, $\mathcal{D} \subseteq \mathcal{D}'$, $\overline{\Gamma} \subseteq \overline{\Gamma}'$ and $\overline{\Delta} \subseteq \overline{\Delta}'$ then $\vDash \mathcal{D}'; \overline{\Gamma}' \supset \overline{\Delta}'$.

Proof. Trivial. □

When combining the general improvement with use-check we have to access the use-sets of the sequents that are known to be valid. From an algorithmic point of view we thus have to store those use-sets together with the path they belong to. We can do this for example with an array that stores a use-set for a path in the proof tree. An example is given in Algorithm 9.

Algorithm 9 is a combination of the algorithms 7 and 8. Its arguments are the labeled residue sequent $\mathcal{D}; \overline{W}, \vec{\overline{R}} \supset \overline{\Delta}$ that is to prove, the current position $\vec{\mathsf{p}}$ in the search tree, the set pos of paths whose sequents are known to be valid and an array $\mathbb{D}$ that holds for each element of pos the corresponding use-set. In the sequent that is to prove we use a vector of labeled residues $\vec{\overline{R}}$ to imply that we have a given order in which we process the residues.

The combination of the two algorithms is more or less straight forward. General improvement is done at the beginning and the end of the algorithm. We thereby get the appropriate use-set from $\mathbb{D}$ if general improvement is successful (lines 2–4) and store newly found use-sets into $\mathbb{D}$ (line 24).

Algorithm 9 Proof search with general improvement and use-check for residue sequents

```
 1: function PRC2PROVABLEG(𝒟; W̄, R⃗ ⊃ Δ̄, p⃗, pos, 𝔻)
 2:     if q⃗ ≤ p⃗ for some q⃗ ∈ pos then
 3:         𝒟 := 𝔻[q⃗]
 4:         return true
 5:     if len(R⃗) = 0 then
 6:         success := CPC2PROVABLE(𝒟; W̄ ⊃ Δ̄)
 7:     else
 8:         let A ⋮^m /C be the first element of R⃗
 9:         let R⃗′ be R⃗ without its first element
10:         success := PRC2PROVABLEG(𝒟; W̄, R⃗′ ⊃ Δ̄, p⃗ ∘ (s), pos, 𝔻)
11:         if not success then
12:             choose a fresh label n ∈ ℕ
13:             success := PRC2PROVABLEG(𝒟_r; W̄, R⃗′, C^n ⊃ Δ̄, p⃗ ∘ (r), pos, 𝔻)
14:             if n ∉ 𝒟_r then
15:                 𝒟 := 𝒟_r
16:             else
17:                 success := PRC2PROVABLEG(𝒟_l; W̄, R⃗′ ⊃ A^n, p⃗ ∘ (l), pos, 𝔻)
18:                 if n ∉ 𝒟_l then
19:                     𝒟 := 𝒟_l
20:                 else
21:                     𝒟 := (𝒟_l ∪ 𝒟_r ∪ {m}) \ {n}
22:     if success then
23:         pos := pos ∪ {p⃗}
24:         𝔻[p⃗] := 𝒟
25:     return success
```

6.2 Proving Residue Sequents Differently

Besides backward proof search we examine two approaches to verify the validity of a residue sequent $W, R \supset \Delta$. Both approaches are based on Theorem 5.7 which says that $\mathrm{Cl}(W, R) = \mathrm{Th}(W \cup \mathrm{con}(\mathrm{GD}(R, \mathrm{Cl}(W, R))))$. This implies that $W, R \supset \Delta$ is valid if $W, \mathrm{con}(\mathrm{GD}(R, \mathrm{Cl}(W, R))) \supset \Delta$ is valid.

In the first approach we compute $\mathrm{Cl}(W, R)$. Since the definition of the closure operator is not suited for its computation we use an alternative, equivalent definition of it.

The second approach takes the succedent that is to be proven into account. There we concentrate on those residues in our theory that are relevant to prove the succedent. That is we search for a subset $R' \subseteq R$ such that $W, R' \supset \Delta$ is valid.

The second approach requires the computation of the so called minimal quasi-supports of a formula on a residue theory. They can be computed with a modified CPC prover. We examine this prover at the end of this section.

6.2.1 Computing $\mathrm{Cl}(W, R)$

$\mathrm{Cl}(W, R)$ is defined iteratively on the sets $\mathrm{Cl}_i(W, R)$. For finite residue theories, it can be shown that the iteration ends after finitely man steps. However, since $\mathrm{Cl}_{i+1}(W, R)$ contains $\mathrm{Th}(\mathrm{Cl}_i(W, R))$, the sets $\mathrm{Th}(\mathrm{Cl}_i(W, R))$ are all infinite for $i > 0$. Computing $\mathrm{Cl}(W, R)$ according to its definition is thus not practicable. In this section we give an alternative, equivalent definition of $\mathrm{Cl}(W, R)$ which is better suited for computations.

Definition 6.20 ($\mathrm{Cl}'(W, R)$)
For a residue theory $\langle W, R\rangle$ we define

$$\mathrm{Cl}'(W, R) := \bigcup_{i=0}^{\infty} \mathrm{Cl}'_i(W, R)$$

where

$$\begin{aligned} \mathrm{Cl}'_0(W, R) &:= W \\ \mathrm{Cl}'_{i+1}(W, R) &:= W \cup \left\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_i(W, R))\right\}. \end{aligned}$$

The definition above is such that $\mathrm{Cl}'(W, R)$ is a subset of $W \cup \mathrm{con}(R)$ and thus finite for finite residue theories.

The following two lemmas are used to show that for finite residue theories $\mathrm{Cl}'(W, R)$ can be calculated in finitely many steps.

Lemma 6.21 (monotonicity of $\mathrm{Cl}'_i(W,R)$)
Let $\langle W,R\rangle$ be a residue theory and $i \in \mathbb{N}$. Then

$$\mathrm{Cl}'_i(W,R) \subseteq \mathrm{Cl}'_{i+1}(W,R).$$

Proof. We show our claim by induction on i.

$i = 0$: Then $\mathrm{Cl}'_0(W,R) = W \subseteq W \cup \{C : A/C \in R \text{ and } A \in \mathrm{Th}(W)\} = \mathrm{Cl}'_1(W,R)$

$i = n+1$: Let $T_n := \mathrm{Th}(\mathrm{Cl}'_n(W,R))$ and $T_{n+1} := \mathrm{Th}(\mathrm{Cl}'_{n+1}(W,R))$. By induction hypothesis we know $\mathrm{Cl}'_n(W,R) \subseteq \mathrm{Cl}'_{n+1}(W,R)$. Hence $T_n \subseteq T_{n+1}$. Now

$$\begin{aligned}\mathrm{Cl}'_{n+1}(W,R) &= W \cup \{\Gamma : A/\Gamma \in R \text{ and } A \in T_n\} \text{ and}\\ \mathrm{Cl}'_{n+2}(W,R) &= W \cup \{\Gamma : A/\Gamma \in R \text{ and } A \in T_{n+1}\}.\end{aligned}$$

Since $T_n \subseteq T_{n+1}$ we know $\mathrm{Cl}'_{n+1}(W,R) \subseteq \mathrm{Cl}'_{n+2}(W,R)$ □

Lemma 6.22 (stagnation in $\mathrm{Cl}'_i(W,R)$)
Let $\langle W,R\rangle$ be a residue theory and $i \in \mathbb{N}$.
If $\mathrm{Cl}'_i(W,R) = \mathrm{Cl}'_{i+1}(W,R)$ then $\mathrm{Cl}'_{i+1}(W,R) = \mathrm{Cl}'_{i+2}(W,R)$.

Proof. Suppose $\mathrm{Cl}'_i(W,R) = \mathrm{Cl}'_{i+1}(W,R)$. Then

$$\begin{aligned}\mathrm{Cl}'_{i+2}(W,R) &= W \cup \left\{\Gamma : A/C \in R \text{ and } A \in \mathrm{Cl}'_{i+1}(W,R)\right\}\\ &= W \cup \left\{\Gamma : A/C \in R \text{ and } A \in \mathrm{Cl}'_{i}(W,R)\right\}\\ &= \mathrm{Cl}'_{i+1}(W,R)\end{aligned}$$

□

The previous two lemmas imply the following corollary.

Corollary 6.23 (maximal $\mathrm{Cl}'_i(W,R)$)
Let $\langle W,R\rangle$ be a residue theory and $i \in \mathbb{N}$. If $\mathrm{Cl}'_i(W,R) = \mathrm{Cl}'_{i+1}(W,R)$ then

1. $\mathrm{Cl}'_i(W,R) = \mathrm{Cl}'_j(W,R)$ for $j \geq i$.
2. $\mathrm{Cl}'(W,R) = \mathrm{Cl}'_i(W,R)$.

$\mathrm{Cl}'(W,R)$ can be computed in finitely many steps if $\langle W,R\rangle$ is finite.

Theorem 6.24 (finite computability of $\mathrm{Cl}'(W,R)$)
Let $\langle W,R\rangle$ be a finite residue theory.
Then there exists $i \leq |R|$ such that $\mathrm{Cl}'(W,R) = \mathrm{Cl}'_i(W,R)$.

Proof. Let $\langle W,R\rangle$ be a finite residue theory, $m = |W|$ and $n = |R|$. We know that $\mathrm{Cl}'_j(W,R) \subseteq W \cup \mathrm{con}(R)$ for all $j \in \mathbb{N}$. Hence $m \leq |\mathrm{Cl}'_j(W,R)| \leq m+n$ for all $j \in \mathbb{N}$. Furthermore we know that $\mathrm{Cl}'_j(W,R) \subseteq \mathrm{Cl}'_{j+1}(W,R)$ for all $j \in$

$\mathbb{N}$. Therefore $m = |\mathrm{Cl}'_0(W,R)| \leq |\mathrm{Cl}'_1(W,R)| \leq \cdots \leq |\mathrm{Cl}'_{n+1}(W,R)| \leq m+n$. There must thus exist $i \leq n$ such that $|\mathrm{Cl}'_i(W,R)| = |\mathrm{Cl}'_{i+1}(W,R)| \leq m+n$. Since $\mathrm{Cl}'_i(W,R) \subseteq \mathrm{Cl}'_{i+1}(W,R)$ we thus know $\mathrm{Cl}'_i(W,R) = \mathrm{Cl}'_{i+1}(W,R)$ from which our claim follows with Corollary 6.23 □

For a residue theory $\langle W,R\rangle$ with infinitely many residues, $\mathrm{Cl}'(W,R)$ may not be computed in finitely many steps. Consider for example $\langle W,R\rangle$ with $W := \{p_0\}$ and $R := \{p_i/p_{i+1} : i \in \mathbb{N}\}$. Then $\mathrm{Cl}'_i(W,R) = \{p_j : 0 \leq j \leq i\}$ and hence $\mathrm{Cl}'_i(W,R) \subsetneq \mathrm{Cl}'_{i+1}(W,R)$ for all $i \in \mathbb{N}$.

It remains to show that $\mathrm{Th}(\mathrm{Cl}'(W,R)) = \mathrm{Cl}(W,R)$. We start by showing that $\mathrm{Cl}(W,R)$ also evolves monotonically.

Lemma 6.25 (monotonicity of $\mathrm{Cl}_i(W,R)$)
Let $\langle W,R\rangle$ be a residue theory and $i \in \mathbb{N}$. Then

$$\mathrm{Cl}_i(W,R) \subseteq \mathrm{Cl}_{i+1}(W,R).$$

Proof. We know that $\mathrm{Cl}_i(W,R) \subseteq \mathrm{Th}(\mathrm{Cl}_i(W,R))$. Furthermore we know that $\mathrm{Th}(\mathrm{Cl}_i(W,R)) \subseteq \mathrm{Cl}_{i+1}(W,R)$. Hence $\mathrm{Cl}_i(W,R) \subseteq \mathrm{Cl}_{i+1}(W,R)$. □

With the previous lemma and Lemma 6.21 we can give alternative definitions of $\mathrm{Cl}(W,R)$ and $\mathrm{Cl}'(W,R)$.

Proposition 6.26 ($\mathrm{Cl}'_i(W,R)$ and $\mathrm{Cl}_i(W,R)$ as limits)
Let $\langle W,R\rangle$ be a residue theory. Then

$$\mathrm{Cl}'(W,R) = \lim_{n\to\infty} \mathrm{Cl}'_n(W,R) \qquad \text{and} \qquad \mathrm{Cl}(W,R) = \lim_{n\to\infty} \mathrm{Cl}_n(W,R)$$

Proof. The claims follow directly from Lemma 6.21 and 6.25 and the definitions of $\mathrm{Cl}'(W,R)$ and $\mathrm{Cl}(W,R)$. □

The following lemma shows how $\mathrm{Cl}_i(W,R)$ and $\mathrm{Cl}'_i(W,R)$ correlate. With the previous proposition we may then conclude the main theorem of this section.

Lemma 6.27 (correlation of $\mathrm{Cl}_i(W,R)$ and $\mathrm{Cl}'_i(W,R)$)
Let $\langle W,R\rangle$ be a residue theory. Then

$$\mathrm{Th}(\mathrm{Cl}_i(W,R)) = \mathrm{Th}(\mathrm{Cl}'_i(W,R))$$

Proof. We show our claim by induction in i.

$i = 0$: This case is clear since $\mathrm{Cl}_0(W,R) = \mathrm{Cl}'_0(W,R) = W$.

$i = n + 1$: We know $\mathrm{Th}(\mathrm{Cl}_n(W,R)) = \mathrm{Th}(\mathrm{Cl}'_n(W,R))$ by induction hypothesis and thus have

$$\begin{aligned}
&\mathrm{Th}(\mathrm{Cl}_{n+1}(W,R)) = \\
&\quad \mathrm{Th}(\mathrm{Th}(\mathrm{Cl}_n(W,R)) \cup \{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}_n(W,R))\}) = \\
&\quad \mathrm{Th}(\mathrm{Th}(\mathrm{Cl}'_n(W,R)) \cup \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_n(W,R))\big\}) = \\
&\qquad\qquad \mathrm{Th}(\mathrm{Cl}'_n(W,R) \cup \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_n(W,R))\big\})
\end{aligned}$$

It is now sufficient to show that

$$\mathrm{Cl}'_n(W,R) \cup \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_n(W,R))\big\} = \mathrm{Cl}'_{n+1}(W,R).$$

For this we distinguish two cases.

$n = 0$: Then

$$\underbrace{\mathrm{Cl}'_n(W,R)}_{=W} \cup \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_n(W,R))\big\} = \mathrm{Cl}'_{n+1}(W,R)$$

$n > 0$: Then we know by definition of $\mathrm{Cl}'_n(W,R)$ that

$$\begin{aligned}
&\mathrm{Cl}'_n(W,R) \cup \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_n(W,R))\big\} = \\
&\qquad W \cup \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_{n-1}(W,R))\big\} \cup \\
&\qquad\qquad \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_n(W,R))\big\}
\end{aligned}$$

Since $\mathrm{Cl}'_{n-1}(W,R) \subseteq \mathrm{Cl}'_n(W,R)$ (Lemma 6.21) this is equal to

$$W \cup \big\{C : A/C \in R \text{ and } A \in \mathrm{Th}(\mathrm{Cl}'_n(W,R))\big\} = \mathrm{Cl}'_{n+1}(W,R). \qquad \square$$

Theorem 6.28 (equality of $\mathrm{Cl}(W,R)$ and $\mathrm{Cl}'(W,R)$)
Let $\langle W,R\rangle$ be a residue theory. Then

$$\mathrm{Cl}(W,R) = \mathrm{Th}(\mathrm{Cl}'(W,R))$$

Proof. The claim follows directly from Lemma 6.27, Proposition 6.26 and Lemma 5.13.3. $\square$

The closure of a finite residue theory is according to Theorem 6.24 and Theorem 6.28 computable in finitely many steps. To verify whether a residue sequent $W, R \supset \Delta$ is valid we hence verify whether the CPC sequent $\mathrm{Cl}'(W,R) \supset \Delta$ is valid.

Algorithm 10 Computing $\mathrm{Cl}'(W, R)$

```
1: function Cl'(W, R)
2:     repeat
3:         R' := ∅
4:         for all δ ∈ R do
5:             if cpcProvable(W ⊃ pre(δ)) then
6:                 R' := R' ∪ {δ}
7:         W := W ∪ con(R')
8:         R := R \ R'
9:     until R' = ∅
10:    return W
```

Two Algorithms to Compute $\mathrm{Cl}'(W, R)$

An algorithm to compute $\mathrm{Cl}'(W, R)$ can easily be derived from its definition (see Algorithm 10).

For better performance we use a slightly modified version (see Algorithm 11). The difference to Algorithm 10 is that we add the consequent of a residue to W as soon as we know that its prerequisite is in $\mathrm{Th}(W)$ (line 7). Consider for example the residue theory $\langle W, R_n \rangle$ with $W := \{p_1\}$ and $R_n := \{p_i/p_{i+1} : 1 \leq i \leq n\}$. Algorithm 10 needs to call the CPC prover $(n^2 + n)/2$ times. The performance of Algorithm 11 depends on the order in which the residues are processed. In the best case it gets along with calling the CPC prover n times. The worst case behavior is equivalent to that of Algorithm 10.

Algorithm 11 Alternative to compute $\mathrm{Cl}'(W, R)$

```
1: function Cl'2(W, R)
2:     repeat
3:         done := true
4:         for all δ ∈ R do
5:             if cpcProvable(W ⊃ pre(δ)) then
6:                 done = false
7:                 W := W ∪ {con(δ)}
8:                 R := R \ {δ}
9:     until done
10:    return W
```

6.2.2 Computing $\mathrm{Cl}(W, R)$ Partially

Given a residue theory $\langle W, R\rangle$ and a formula A, it is often sufficient to verify for a subset $R' \subsetneq R$ whether $A \in \mathrm{Cl}(W, R')$.

Consider the residue theory $\langle W, R_n\rangle$ where $R_n := \{p_i/p_{i+1} : 0 \leq i \leq n\}$ and $W := \{p_1\}$. Then $p_2 \in \mathrm{Cl}(W, R_n)$ for $n \geq 2$.
According to Lemma 5.13 $\mathrm{Cl}(W, R_n)$ is a subset of $\mathrm{Cl}(W, R_m)$ if $n \leq m$. To verify for $n \geq 2$ whether p_2 is in $\mathrm{Cl}(W, R_n)$ it is thus sufficient to verify whether p_2 is in $\mathrm{Cl}(W, R_2)$. The advantage is obvious. Algorithm 11 would have to call the function CPCPROVABLE between 10 and 55 times to compute $\mathrm{Cl}'(W, R_{10})$ and only 2 or 3 times to compute $\mathrm{Cl}'(W, R_2)$.

In this section we investigate an approach with which such redundancies may be avoided. We illustrate it first on a simple example.

Introductory Example

Let $W := \{p_1, p_2\}$, $R := \left\{\frac{p_1}{q_1}, \frac{p_2}{q_2}, \frac{s}{r}, \frac{q_1 \vee q_2}{r}, \frac{q_2}{t}\right\}$ and suppose that we want to verify whether $r \in \mathrm{Th}(\mathrm{Cl}'(W, R))$, i.e. whether $W, R \supset r$ is valid.

In our approach we concentrate on the consequents of the residues. That is instead of calculating $\mathrm{Cl}'(W, R)$ we check the validity of $W, \mathrm{con}(R) \supset r$. If this fails, then we know that the residue sequent is not valid, otherwise we have to take further steps to decide its validity.
In our example $W, \mathrm{con}(R) \supset r$ is valid. With some additional work in the CPC prover we can calculate those subsets of $\mathrm{con}(R)$ that are minimally needed in order for the sequent to be valid. In our example there is just one such subset: $\{r\}$. Now r can either originate from s/r or $q_1 \vee q_2/r$, i.e. one of those residues is crucial for the proof of the residue sequent. Hence the prerequisite of at least one of those residues must be in $\mathrm{Cl}(W, R)$. This is equivalent to $s \in \mathrm{Cl}(W, R \setminus \{s/r\})$ or $q_1 \vee q_2 \in \mathrm{Cl}(W, R \setminus \{q_1 \vee q_2/r\})$ (Lemma 5.14).

We consider the first choice and proceed as before. Let $R_1 := R \setminus \{s/r\}$. Proving $W, \mathrm{con}(R_1) \supset s$ fails, thus $s \notin \mathrm{Th}(W \cup \mathrm{con}(R_1))$. Since $\mathrm{Cl}(W, R_1) \subseteq \mathrm{Th}(W \cup \mathrm{con}(R_1))$ we know $s \notin \mathrm{Cl}(W, R_1)$.
Hence the second choice remains and we let $R_2 := R \setminus \{q_1 \vee q_2/r\}$. Proving $W, \mathrm{con}(R_2) \supset q_1 \vee q_2$ succeeds and p_1/q_1 or p_2/q_2 turn out to be crucial for its validity.

Again we consider the first choice and proceed as before. Let $R_{21} := R_2 \setminus \{p/q_1\}$. Proving $W, \mathrm{con}(R_{21}) \supset p$ succeeds and no residue of R_{21} is crucial for its proof. Considering the second choice is thus not necessary and we know that $p \in \mathrm{Cl}(W, \{p/q_1, q_1 \vee q_2/r\}) \subseteq \mathrm{Cl}(W, R)$.

Proving ends here and we hence know that $W, R \supset r$ is valid.

Quasi-Supports

In this section we investigate our approach formally. A central point in the introductory example is to identify those sets of residues, whose conclusions are crucial for the validity of the CPC sequents. Formally they correspond to so called quasi-supports.

Definition 6.29 (quasi-support, support)
Let $\langle W, R\rangle$ be a residue theory and A a formula.

R' is called a *quasi-support* of A for $\langle W, R\rangle$ if $A \in \mathrm{Th}(W \cup \mathrm{con}(R'))$ and $R' \subseteq R$.
It is called *minimal* if $A \notin \mathrm{Th}(W \cup \mathrm{con}(R''))$ for all $R'' \subsetneq R'$.

R' is called a *support* of A for $\langle W, R\rangle$ if $A \in \mathrm{Cl}(W, R')$ and $R' \subseteq R$.
It is called *minimal* if $A \notin \mathrm{Cl}(W, R'')$ for all $R'' \subsetneq R'$.

We write $\mathrm{minqs}(A, \langle W, R\rangle)$ to denote the minimal quasi-supports of A for $\langle W, R\rangle$.

Given a residue theory $\langle W, R\rangle$ and $\delta \in R$ we often say that δ has a support or is supported (in $\langle W, R\rangle$) if there exists a support of $\mathrm{pre}(\delta)$ for $\langle W, R \setminus \{\delta\}\rangle$.

Remark 6.30 (quasi-support vs. support)
If R' is a quasi-support of A for $\langle W, R\rangle$ then this does not imply that $A \in \mathrm{Cl}(W, R')$. However, if R' is a support of A for $\langle W, R\rangle$ then this implication holds (Lemma 5.13.1). Consider the introductory example. There $R' := \{s/r\}$ is a quasi-support of r for $\langle W, R\rangle$ but $r \notin \mathrm{Cl}(W, R')$.

A support of A for $\langle W, R\rangle$ is always also a quasi-support of it. If the empty set is a quasi-support of A for $\langle W, R\rangle$ then it is also a support of it.

Remark 6.31 (use-set as support)
Given a valid labeled default sequent $\mathcal{D}; \overline{W}, \overline{R} \supset \overset{\mathrm{n}}{A}$ it is easy to see that $\overline{R}{\downarrow_{\mathcal{D}}}$ corresponds to a support of A for $\langle W, R\rangle$, where A, W and R are the label-less versions of $\overset{\mathrm{n}}{A}$, $\overline{W}$ and $\overline{R}$.

The following lemma states that the prerequisites and conclusions of a minimal support of A for $\langle W, R\rangle$ are in $\mathrm{Cl}(W, R)$.

Lemma 6.32 (minimal support as closed subtheory)
Let R' be a minimal support of A for $\langle W, R\rangle$. Then

1. $\mathrm{pre}(\delta) \in \mathrm{Cl}(W, R)$ for all $\delta \in R'$.
2. $\mathrm{con}(\delta) \in \mathrm{Cl}(W, R)$ for all $\delta \in R'$.

Proof. Let R' be a minimal support of A for $\langle W, R\rangle$ $(*)$.
For $R' = \emptyset$ the claim follows immediately. So suppose that $R' \neq \emptyset$.

1. Let $\delta \in R'$ and suppose that $\mathrm{pre}(\delta) \notin \mathrm{Cl}(W, R)$, hence $\mathrm{pre}(\delta) \notin \mathrm{Cl}(W, R')$. Then we know by Lemma 5.14.2 that $\mathrm{Cl}(W, R') = \mathrm{Cl}(W, R' \setminus \{\delta\})$. This contradicts to $(*)$, hence $\mathrm{pre}(\delta) \in \mathrm{Cl}(W, R)$.

2. This claim follows immediately from the first claim and Lemma 5.13.4. □

The following proposition is central for our approach. It ensures that investigating the minimal quasi-supports is sufficient to verify validity.

Proposition 6.33 (existence of extensible minimal quasi-support)
Let $\langle W, R\rangle$ be a residue theory and A a formula.

If $A \in \mathrm{Cl}(W, R)$ then there exists $R' \in \mathrm{minqs}(A, \langle W, R\rangle)$ such that $\mathrm{pre}(\delta) \in \mathrm{Cl}(W, R \setminus \{\delta\})$ for all $\delta \in R'$.

Proof. Suppose that $A \in \mathrm{Cl}(W, R)$. Then there exists a minimal support of A for $\langle W, R\rangle$. Let R'' be such a minimal support $(*)$ and let R' be a minimal quasi-support of A for $\langle W, R''\rangle$. Then R' is also a minimal quasi-support of A for $\langle W, R\rangle$. From $(*)$ we know by the previous Lemma that $\mathrm{pre}(\delta) \in \mathrm{Cl}(W, R'') \subseteq \mathrm{Cl}(W, R)$ for all $\delta \in R'' \supseteq R'$. Hence $\mathrm{pre}(\delta) \in \mathrm{Cl}(W, R)$ for all $\delta \in R'$. Our claim follows with Lemma 5.14.4. □

The following proposition ensures that a quasi-support can be extended to a support if it fulfills the properties mentioned in the previous proposition.

Proposition 6.34 (extensible quasi-support)
Let R' be a quasi-support of A on $\langle W, R\rangle$.

If $\mathrm{pre}(\delta) \in \mathrm{Cl}(W, R \setminus \{\delta\})$ for all $\delta \in R'$, then $A \in \mathrm{Cl}(W, R)$.

Proof. Let R' be a quasi-support of A for $\langle W, R\rangle$ (1) and suppose that $\mathrm{pre}(\delta) \in \mathrm{Cl}(W, R \setminus \{\delta\})$ for all $\delta \in R'$. Then we know by Lemma 5.13.4 that $\mathrm{con}(\delta) \in \mathrm{Cl}(W, R \setminus \{\delta\})$ for all $\delta \in R'$. Since $\mathrm{Cl}(W, R \setminus \{\delta\}) \subseteq \mathrm{Cl}(W, R)$ (Lemma 5.13.1) and $W \subseteq \mathrm{Cl}(W, R)$ (Lemma 5.13.2) we hence know $W \cup \mathrm{con}(R') \subseteq \mathrm{Cl}(W, R)$ (2). From (1) we know $A \in \mathrm{Th}(W \cup \mathrm{con}(R'))$ and from (2) and Lemma 5.13.3 we obtain $\mathrm{Th}(W \cup \mathrm{con}(R')) \subseteq \mathrm{Th}(\mathrm{Cl}(W, R)) \subseteq \mathrm{Cl}(W, R)$. Hence $A \in \mathrm{Cl}(W, R)$. □

From the previous proposition we can easily derive an algorithm to verify whether A is in $\mathrm{Cl}(W, R)$.

We calculate the minimal quasi-supports of A for $\langle W, R\rangle$ and try whether one of them can be extended to a support of A. This can be done recursively. Given a minimal quasi-support R' of A for $\langle W, R\rangle$ we try for each residue

$\delta \in R'$ whether one of the quasi-supports of $\mathrm{pre}(\delta)$ for $\langle W, R \setminus \{\delta\}\rangle$ can be extended to a support of it. The recursion ends if no minimal quasi-support is found or if the empty set is the one and only minimal quasi-support. The pseudocode is given in Algorithm 12.

Algorithm 12 Using minimal quasi-supports to prove residue sequents

```
1:  function prcProvableQS(⟨W, R⟩ , A)
2:      R := minQuasiSupports(⟨W, R⟩ , A)
3:      success := false
4:      while not success and R ≠ ∅ do
5:          select R_mqs ∈ R and let R := R \ {R_mqs}
6:          success := true
7:          for all δ ∈ R_mqs do
8:              R' := R_mqs \ {δ}
9:              A' := pre(δ)
10:             success := success and prcProvableQS(⟨W, R'⟩ , A')
11:     return success
```

Remark 6.35 (calculate support found by algorithm)
On success prcProvableQS does not supply us with the support that it found. But it is easy to obtain the support. We can remember the quasi-supports R_{mqs} for which the functions calls were successful, join them on success and obtain like that a support for the proven sequent.

We are left with the problem to compute the minimal quasi-supports of A for $\langle R, W\rangle$. Before we investigate this problem we show a variant of the given algorithm in which the results about the minimal quasi-supports can be reused.

Caching Intermediate Results

Algorithm 12 contains certain redundancies. In its recursive calls we calculate the minimal quasi-supports of A' for $\langle W, R'\rangle$, where A' is a prerequisite of some residue $\delta \in R$ and $R' \subsetneq R$. Thereby it is possible that we encounter in different recursive calls the same prerequisite A' together with different subsets R' and R'' of R.

If $R'' \subseteq R'$ then we can compute the minimal quasi-supports of A' for $\langle W, R''\rangle$ from the minimal quasi-supports of A' for $\langle W, R'\rangle$.

Proposition 6.36 (minimal quasi-support in subtheories)
Let $\langle W, R\rangle$ be a residue theory and $R' \subseteq R$.

1. If R_A is a minimal quasi-support of A for $\langle W, R'\rangle$ then R_A is a minimal quasi-support of A for $\langle W, R\rangle$.

2. If R_A is a minimal quasi-support of A for $\langle W, R\rangle$ and $R_A \subseteq R'$ then R_A is a minimal quasi-support of A for $\langle W, R'\rangle$.

Proof. The claim follows from the definition of a minimal quasi-support. □

Hence, if $\mathcal{R}$ is the set of quasi-supports of A for $\langle W, R'\rangle$ and $R'' \subseteq R'$, then $\{R_A \in \mathcal{R} : R_A \subseteq R''\}$ is the set of quasi-supports of A for $\langle W, R'\rangle$.

We illustrate this on the following example.

$$W := \{q_1 \vee q_3 \to p_1, q_1 \to p_2, p_3, q_1 \wedge q_2 \to p\} \quad R := \left\{\frac{p_1}{q_1}, \frac{p_2}{q_2}, \frac{p_3}{q_3}\right\} \quad A := p$$

A possible run of Algorithm 12 could result in calculating the minimal quasi-supports as follows.

minQuasiSupports$(\langle W, \{p_1/q_1, p_2/q_2, p_3/q_3\}\rangle, p) = \{\{p_1/q_1, p_2/q_2\}\}$
One minimal quasi-support, start with p_1/q_1.
1 minQuasiSupports$(\langle W, \{p_2/q_2, p_3/q_3\}\rangle, p_1) = \{\{p_2/q_2\}, \{p_3/q_3\}\}$
Two minimal quasi-supports. Continue with first.
1.1 a minQuasiSupports$(\langle W, \{p_3/q_3\}\rangle, p_2) = \emptyset$
No success, try second quasi-support.
1.1 b minQuasiSupports$(\langle W, \{p_2/q_2\}\rangle, p_3) = \{\emptyset\}$
Success for p_1/q_1, continue with p_2/q_2.
2 minQuasiSupports$(\langle W, \{p_1/q_1, p_3/q_3\}\rangle, p_2) = \{\{p_1/q_1\}\}$
2.1 minQuasiSupports$(\langle W, \{p_3/q_3\}\rangle, p_1) = \{\{p_3/q_3\}\}$
2.1.1 minQuasiSupports$(\langle W, \emptyset\rangle, p_3) = \{\emptyset\}$

At 1 we compute the minimal quasi-supports of p_1 for $\langle W, \{p_2/q_2, p_3/q_3\}\rangle$ and obtain $\{\{p_2/q_2\}, \{p_3/q_3\}\}$. Later at 2.1 we compute the minimal quasi-supports of p_1 for $\langle W, \{p_3/q_3\}\rangle$ and obtain $\{\{p_3/q_3\}\}$ which is a subset of the previously computed one. In this case we can compute the latter set from the former one.

At 1.1 we compute the minimal quasi-supports of p_2 for $\langle W, \{p_3/q_3\}\rangle$ and obtain an empty set. Later at 2 we compute the minimal quasi-supports of p_2 for $\langle W, \{p_1/q_1, p_3/q_3\}\rangle$ and obtain $\{\{p_3/q_3\}\}$. In this case the proposition is of no use because $\{p_3/q_3\}$ is not a superset of $\{p_1/q_1, p_3/q_3\}$.

To reuse the information about minimal quasi-supports in Algorithm 12 we can store the calculated quasi-supports together with the corresponding residues. Then before calculating the minimal quasi-supports of A for $\langle W, R\rangle$ we check whether we have already calculated them for a superset of R. If this is the case, then we calculate the minimal quasi-supports according to the previous proposition.

Now instead of calculating the minimal quasi-supports of pre(δ) for some subset R' of the original set of residues R, we can calculate them directly for the maximal subset of R in question, i.e. for $R \setminus \{\delta\}$. Like this we do not have to check whether we have calculated them for a superset of the current set of residues already, but only whether we have calculated them already or not. This strategy turned out to be very useful for the residue sequents we have tested. A corresponding algorithm is given in Algorithm 13.

prcProvableQSC($\langle W, R\rangle, A$) This function initiates proof search. It creates the global map MQS[] that maps from a residue to its calculated minimal quasi-supports, calculates the minimal quasi-supports of the given residue sequent and starts proving by calling prcProvableQSC($\langle W, R\rangle$, R_{out}, $\mathcal{R}$).

prcProvableQSC($\langle W, R\rangle$, R_{OUT}, $\mathcal{R}$) This function does recursive proof search. Its arguments are a residue theory $\langle W, R\rangle$, a set of excluded residues $R_{\text{out}} \subseteq R$ and a set of minimal quasi-supports $\mathcal{R}$. The function checks whether an element of $\mathcal{R}$ is extensible as follows.
It loops over the given set of minimal quasi-supports (lines 7–14) and recursively tries to extend one of them to a support (lines 10–15). Thereby it computes the minimal quasi-supports for a residue only if needed (lines 11–12) and calculates the minimal quasi-supports that fits to $R \setminus R_{\text{out}}$ from the cached results (line 13).

Avoiding Superfluous Loops

When testing residue sequents that had a lot of circular dependencies in the antecedent, our prover sometimes started to loop without calculating new minimal quasi-supports. This looping had two reasons.

First, when selecting R' on line 8, it often selected a minimal quasi-support that contained only elements for which it had already cached the minimal quasi-supports. To avoid this we prioritize in that selection those minimal quasi-supports $\mathcal{R}$ that contain open residues, i.e. residues for which we have not yet calculated the minimal quasi-supports. Furthermore we prioritize open residues in the loop on lines 10–14.

Second, when trying to prove invalid sequents the prover reached a point where no more minimal quasi-supports could be calculated, i.e. where we have reached a saturation of necessary information. Nevertheless it kept on backtracking and looping over the minimal quasi-supports. To avoid this we use a method to detect whether all relevant minimal quasi-supports have been calculated and stop the proof search if this is the case.
The method uses a global set of residues R_{open} that contains those residues that we encountered in the proof search algorithm but for which we did not

try to calculate a support. Adding items to R_{open} thereby happens in two cases.

1. Extending a minimal quasi-support R' to a support fails and not all residues in R' were processed.

   Consider the residue theory $\langle \{p\}, \{q/q_1, p/p_1, p_1/p_2\} \rangle$ and the minimal quasi-support $R_1 = \{q/q_1, p_1/p_2\}$.
   Finding a support of q/q_1 fails, therefore we do not try to find a support of p_1/p_2, the second element of R_1. We hence add it to R_{open}.

2. Extending a minimal quasi-support R' to a support succeeds and there are still minimal-quasi supports left that we can try to extend to a support.

   Consider the same residue theory as above and the minimal quasi-supports $\mathcal{R} = \{R_1, R_2\}$ with $R_1 = \{p_1/p_2\}$ and $R_2 = \{q/q_1, p/p_1\}$.
   Extending R_1 to a support succeeds and in that process we calculate the minimal quasi-supports of p/p_1. Because R_1 could be extended to a support we do not need to process R_2. Nevertheless it contains yet unprocessed residues that may be relevant for the overall proof. We therefore add the elements of R_2 for which we have not yet calculated the minimal-quasi supports to R_{open}. In our example this is q/q_1.

Loop detection is done before we try to extend a minimal quasi-support R' to a support. If there R_{open} is empty and R' contains only elements for which we have already calculated the minimal quasi-supports, we know that there are no more relevant residues left to extend R' to a support.
Now if we stop proof search in such a situation it may be that a valid sequent is not yet verified as being valid. Consider the following example.

$$W := \{p\} \qquad R := \left\{ \frac{p}{p_1}, \frac{p_1}{p_2}, \frac{p_2}{p_3}, \frac{p_3}{p_4} \right\} \qquad A := p_2 \wedge p_4$$

Then $\text{minqs}(A, \langle W, R \rangle) = \{\{p_1/p_2, p_3/p_4\}\}$. If we start to process p_1/p_2 first then the algorithm runs as sketched in Figure 6.5.

The columns in Figure 6.5 contain the recursion level, the list of excluded residues where the currently selected residue is highlighted in bold face, the minimal quasi-supports and the keys of the elements stored in MQS before the recursion. R_{open} is not listed because it remains empty.
The algorithm succeeds in finding a support of p_1/p_2. Then at recursion level 1.2 it starts to search for a support of p_3/p_4. At level 1.2.1 we add the last remaining residue to MQS and hence we have a saturation of information at level 1.2.1.1. We therefore stop proof search and thus skip verifying whether p/p_1 has a support.
In this case we know that the residue has a support (recursion level 1.1.1).

Algorithm 13 Using cached minimal quasi-supports to prove residue sequents

```
1: function prcProvableQSC(⟨W, R⟩, A)
2:     create the global map MQS[]
3:     R := minQuasiSupports(⟨W, R⟩, A)
4:     return prcProvableQSC(⟨W, R⟩, ∅, R)

5: function prcProvableQSC(⟨W, R⟩, R_out, R)
6:     result := false
7:     while not result and R ≠ ∅ do
8:         select R' ∈ R and let R := R \ {R'}
9:         result := true
10:        for all δ ∈ R' do
11:            if not exists MQS[δ] then
12:                MQS[δ] := minQuasiSupports(⟨W, R \ {δ}⟩, pre(δ))
13:            R_δ := {R_δ ∈ MQS[δ] : R_δ ∩ R_out = ∅}
14:            result := result and
15:                      prcProvableQSC(⟨W, R⟩, R_out ∪ {δ}, R_δ)
16:    return result
```

	R_{out}	$\mathcal{R}$	keys(MQS)
1.	$\emptyset$	$\left\{\left\{\frac{p_1}{p_2}, \frac{p_3}{p_4}\right\}\right\}$	$\emptyset$
1.1	$\left\{\frac{\mathbf{p_1}}{\mathbf{p_2}}\right\}$	$\left\{\left\{\frac{p}{p_1}\right\}\right\}$	$\left\{\frac{p_1}{p_2}\right\}$
1.1.1	$\left\{\frac{\mathbf{p}}{\mathbf{p_1}}, \frac{p_1}{p_2}\right\}$	$\{\emptyset\}$	$\left\{\frac{p}{p_1}, \frac{p_1}{p_2}\right\}$
1.2	$\left\{\frac{\mathbf{p_3}}{\mathbf{p_4}}\right\}$	$\left\{\left\{\frac{p_2}{p_3}\right\}\right\}$	$\left\{\frac{p}{p_1}, \frac{p_1}{p_2}, \frac{p_3}{p_4}\right\}$
1.2.1	$\left\{\frac{\mathbf{p_2}}{\mathbf{p_3}}, \frac{p_3}{p_4}\right\}$	$\left\{\left\{\frac{p_1}{p_2}\right\}\right\}$	$\left\{\frac{p}{p_1}, \frac{p_1}{p_2}, \frac{p_2}{p_3}, \frac{p_3}{p_4}\right\}$
1.2.1.1	$\left\{\frac{\mathbf{p_1}}{\mathbf{p_2}}, \frac{p_2}{p_3}, \frac{p_3}{p_4}\right\}$	$\left\{\left\{\frac{p}{p_1}\right\}\right\}$	$\rightarrow$ saturation

Figure 6.5: Example with successful loop check

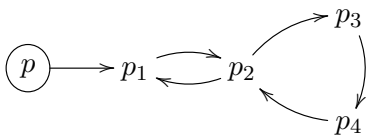
To detect this we keep record of those residues for which a support was found in a set R_{sprt}. In this example we put p/p_1 into R_{sprt} at level 1.1.1. As a consequence p_1/p_2 turns out to have a support, we also put it into R_{sprt}. Hence recursion level 1.2.1.1 can be verified to be successful and the proof succeeds.

The mechanism with R_{sprt} is not sufficient to decide provability on loop detection. Consider the following example.

$$W := \{p\} \qquad R := \left\{ \frac{p}{p_1}, \frac{p_1}{p_2}, \frac{p_2}{p_1}, \frac{p_2}{p_3}, \frac{p_3}{p_4}, \frac{p_4}{p_2} \right\} \qquad A := p_2 \wedge p_4$$

The minimal quasi-supports of A for $\langle W, R\rangle$ are $\left\{ \left\{ \frac{p_1}{p_2}, \frac{p_3}{p_4} \right\}, \left\{ \frac{p_4}{p_2}, \frac{p_3}{p_4} \right\} \right\}$.

We can depict the theory as a graph. A residue A/B is represented by an arrow from A to B, formulas of the base theory are encircled.



In this simple example proof search can be regarded as finding a way back from p_2 to p and from p_4 to p without visiting an arrow more than once. The situation we would like to point out is where we first go back along the path p_2–p_1–p_2–p_4–p_3–p_2. There we detect a loop, backtrack to p_1 and succeed in going the alternative to p. Because of the extra loop all minimal quasi-supports are calculated. Furthermore we know only that p/p_1 and p_1/p_2 have a support, that is $R_{\text{sprt}} = \{p/p, p_1/p_2\}$. Now if we try to find a way back from p_4 to p we have information saturation. Deciding according to R_{sprt} that p_3/p_4 has a support fails because the only minimal support of p_3/p_4 is $\{p_2/p_3\}$ and its element is not in R_{sprt}. With the second minimal quasi-support of A we run into the same problem if we start with searching for a support of p_4/p_2. This situation is given in Figure 6.6. The minimal quasi-supports in the third column that are canceled are those that have been filtered according to R_{out}.

To decide whether a residue has a support in such a situation we can compute the residues that have support from the calculated minimal quasi-supports. The procedure is similar to calculating $\text{Cl}'_n(W, R)$. Given the map of minimal quasi-supports MQS we start with the set $\text{Sprt}_0(\text{MQS})$ of residues for which $\text{MQS}[\delta]$ is $\{\emptyset\}$. These are the residues whose prerequisites are valid in the base theory. Then we iteratively calculate the other residues that have support according to the following schema.

$$\begin{aligned} \text{Sprt}_0(\text{MQS}) &:= \{\delta : \text{MQS}[\delta] = \{\emptyset\}\} \\ \text{Sprt}_{i+1}(\text{MQS}) &:= \{\delta : R \in \text{Sprt}_i(\text{MQS}) \text{ for some } R \in \text{MQS}[\delta]\} \end{aligned}$$

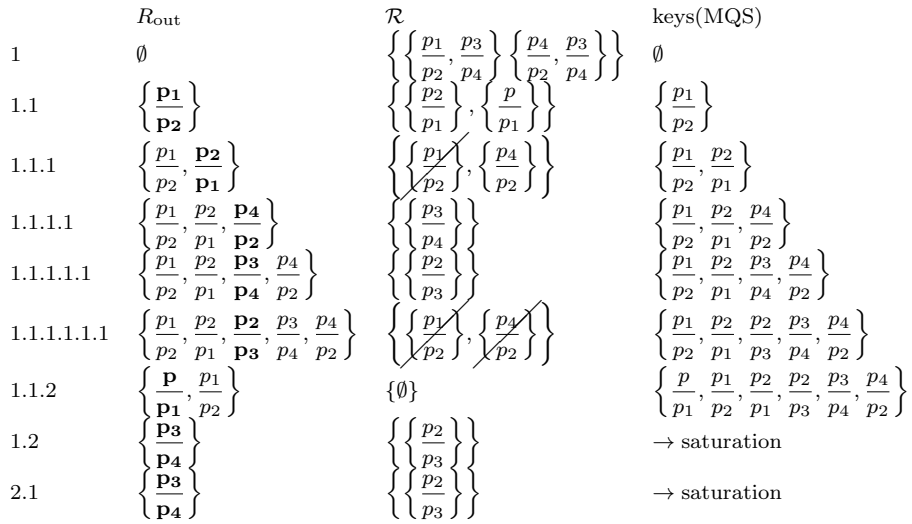


Figure 6.6: Example with successful loop check

We can stop the iteration if no new residues turn out to have a support, that is if $\text{Sprt}_{i+1}(\text{MQS}) = \text{Sprt}_i(\text{MQS})$.

The improved algorithm is given in Algorithm 14.

prcProvableQSCL($\langle W, R\rangle$, A) initiates proof search. It calculates the minimal quasi-supports of A for $\langle W, R\rangle$, initializes the global variables and starts proof search by calling prcProvableQSCL($\langle W, R\rangle$, R_{OUT}, $\mathcal{R}$)

prcProvableQSCL($\langle W, R\rangle$, R_{OUT}, $\mathcal{R}$) does proof search based on a given set of minimal quasi-supports $\mathcal{R}$. Besides the residue theory and $\mathcal{R}$ it takes the set of excluded residues R_{out} as arguments. This argument is passed unmodified to prcProvableQSCL($\langle W, R\rangle$, R_{OUT}, R').
The function first checks whether one of the given minimal quasi-supports is already known to be supported (line 8). If not it checks for each element of $\mathcal{R}$ whether it can be extended to a support (lines 9–11). When returning from the recursive calls it updated the set of open residues accordingly (line 12).

prcProvableQSCL($\langle W, R\rangle$, R_{OUT}, R') tries to extend the given minimal quasi-support R' to a support taking the set of excluded residues R_{out} into account.
The function first does the loop check (lines 15–17). If there are no open residues and the minimal quasi-supports of all residues in R' have been calculated it updates the set of supported residues by calling

Algorithm 14 Using cached minimal quasi-supports and loop check to prove residue sequents

```
1: function prcProvableQSCL(⟨W, R⟩, A)
2:     R := minQuasiSupports(⟨W, R⟩, A)
3:     clear the global map MQS[] of minimal quasi-supports
4:     clear the global set R_sprt of supported residues
5:     clear the global set R_open of open residues
6:     return prcProvableQSCL(⟨W, R⟩, ∅, R)

7: function prcProvableQSCL(⟨W, R⟩, R_out, R)
8:     if {R' ∈ R : R ⊆ R_sprt} ≠ ∅ then return true
9:     for all R' ∈ R do [a]
10:        result := prcProvableQSCL(⟨W, R⟩, R_out, R')
11:        if result then break
12:    R_open := R_open ∪ {δ ∈ ⋃R : δ ∉ keys(MQS)}
13:    return result

14: function prcProvableQSCL(⟨W, R⟩, R_out, R')
15:    if R_open = ∅ and {δ ∈ R' : δ ∉ keys(MQS)} = ∅ then
16:        updateSupported
17:        return R' ⊆ R_sprt
18:    result := true
19:    for all δ ∈ R' do [b]
20:        if δ ∉ keys(MQS) then
21:            MQS[δ] := minQuasiSupports(⟨W, R \ {δ}⟩, pre(δ))
22:        R_δ := {R_δ ∈ MQS[δ] : R_δ ∩ R_out = ∅}
23:        result := prcProvableQSCL(⟨W, R⟩, R_out ∪ {δ}, R_δ)
24:        if result then R_sprt := R_sprt ∪ {δ}
25:        else break
26:    return result

27: function updateSupported
28:    repeat
29:        for all δ ∈ keys(MQS) do
30:            if {R' ∈ MQS[δ] : R' ⊆ R_sprt} ≠ ∅ then R_sprt := R_sprt ∪ {δ}
31:    until R_sprt was not extended anymore
```

[a] prefer R' that has open residues
[b] prefer $\delta \in R_{\text{open}}$

updateSupported (line 16) and returns with success if R' holds only supported residues (line 17).
If the loop check is not successful it checks whether all residues in R' have a support (lines 19–25) and adds those that turn out to have a support to R_{sprt} (line 24).

UPDATESUPPORTED updates the set of supported residues according to the schema discussed above.

Further Improvements

To improve loop check we can remove the residues that are checked on the current recursion level from the set of open residues, that is we let $R_{\text{open}} := R_{\text{open}} \setminus \{\delta \in R' : \delta \notin \text{keys(MQS)}\}$ before checking the residues (line 19). Like this we may obtain information saturation earlier in the search tree.

We can reuse the information that a residue has no support on the current recursion level. Suppose that the set of minimal quasi-supports passed to PRCPROVABLEQSCL($\langle W, R\rangle$, R_{out}, $\mathcal{R}$) contains n minimal quasi-supports, that is $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ and that when trying to extend R_1 it turns out that $\delta \in R_1$ has no support. Then δ will have no support in any attempt to extend the remaining minimal quasi-supports $R_2, \ldots, R_n$ to a support. We can therefore mark δ as unsupported and use that information later to prevent redundant proving. It is important to unmark δ if we leave the recursion level in which it was marked as unsupported (line 13) because afterward R_{out} is no longer a superset of this recursion level's set of excluded residues and therefore δ is no longer known to have no support.

6.2.3 Computing Minimal Quasi-Supports

When discussing the introductory example on page 152 we mentioned that the minimal quasi-supports of A for $\langle W, R\rangle$ can be computed if some additional work is done in the CPC prover. In this section we investigate this approach.

Let us first revisit the definition of a minimal quasi-support. $R' \subseteq R$ is a minimal quasi-support of A for $\langle W, R\rangle$ if $W, \text{con}(R') \supset A$ is valid and $W, \text{con}(R'') \supset A$ is not valid for all proper subsets of R'.
Hence the set $\Gamma' := \text{con}(R')$ has the property that $W, \Gamma' \supset A$ is valid and that $W, \Gamma'' \supset A$ is not valid for any $\Gamma'' \subsetneq \Gamma'$. We say that Γ' is a *minimal enlargement of W for A*.

The idea we follow is to find all subsets of $\mathrm{con}(R)$ that are minimal enlargements of W for A. From each such subset we then identify the corresponding minimal quasi-support. Consider the following example:

$$W := \{p_1, p_2, p_3\} \qquad R := \left\{\frac{p_1}{q_1}, \frac{p_2}{q_2}, \frac{p_3}{q_3}\right\} \qquad A := q_1 \vee q_2 \wedge q_3.$$

The subsets of $\mathrm{con}(R)$ that are minimal enlargements of W for A are $\Gamma_1 := \{q_1\}$ and $\Gamma_2 := \{q_2, q_3\}$. Their corresponding minimal quasi-supports are $R_1 := \{p_1/q_1\}$ and $R_2 := \{p_2/q_2, p_3/q_3\}$.

Now different residues may have the same consequent. For a formula $A \in \mathrm{con}(R)$ it is thus in general not clear from which residue it derives. We therefore label our residues consecutively with natural numbers and pass their labels to their consequents when calculating $\mathrm{con}(R)$. Consider the following example:

$$\overline{W} := \left\{\overset{0}{p_1}, \overset{0}{p_2}\right\} \qquad \overline{R} := \left\{\frac{p_1 \overset{1}{\vdots}}{q_1}, \frac{p_2 \overset{2}{\vdots}}{q_2}, \frac{p_3 \overset{3}{\vdots}}{q_2}\right\} \qquad \overset{0}{A} := q_1 \overset{0}{\wedge} q_2.$$

Here the quasi-supports are $\left\{p_1 \overset{1}{\vdots}/q_1, p_2 \overset{2}{\vdots}/q_2\right\}$ and $\left\{p_1 \overset{1}{\vdots}/q_1, p_3 \overset{3}{\vdots}/q_2\right\}$. When calculating $\mathrm{con}(\overline{R})$ we label the consequents with the label of the corresponding residue and obtain $\mathrm{con}(\overline{R}) = \left\{\overset{1}{q_1}, \overset{2}{q_2}, \overset{3}{q_2}\right\}$.
Because we distinguish between $\overset{2}{q_2}$ and $\overset{3}{q_2}$ there are two minimal enlargements of $\overline{W}$ for $\overset{0}{A}$: $\left\{\overset{1}{q_1}, \overset{2}{q_2}\right\}$ and $\left\{\overset{1}{q_1}, \overset{3}{q_2}\right\}$. They represent the minimal quasi-supports $\left\{p_1 \overset{1}{\vdots}/q_1, p_2 \overset{2}{\vdots}/q_2\right\}$ and $\left\{p_1 \overset{1}{\vdots}/q_1, p_3 \overset{3}{\vdots}/q_2\right\}$.

We also use the labels to distinguish between formulas from W and $\mathrm{con}(R)$. In the example above formulas carrying label 0 are from W all other formulas are from $\mathrm{con}(R)$.

The following definition formalizes this distinction.

Definition 6.37 (L-sequent)
An *L-sequent* is a quadruple $\langle L, \mathcal{D}, \overline{\Gamma}, \overline{\Delta}\rangle$ denoted by $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ where $\overline{\Gamma}$ and $\overline{\Delta}$ are finite multisets of labeled formulas, L is a finite subset of $\mathbb{N}$ and $\mathcal{D} \subseteq L \cap \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$.

We call L the *partitioner* and $\mathcal{D}$ the *filter* of the L-sequent. $\overline{\Gamma}$ and $\overline{\Delta}$ are called *antecedent* and *succedent*.

For an L-sequent $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ we define its *filtered sequent* as

$$\overline{\Gamma}{\downarrow_{L^c}}, \overline{\Gamma}{\downarrow_{\mathcal{D}\cap L}} \supset \overline{\Delta}{\downarrow_{L^c}}, \overline{\Delta}{\downarrow_{\mathcal{D}\cap L}},$$

where L^c denotes the complement of L, i.e. $L^c := \mathbb{N} \setminus L$.

Definition 6.38 (length of an L-sequent)
The length of an L-sequent is defined to be the sum of the length of its formulas.

$$\mathrm{len}(L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}) := \mathrm{len}(\overline{\Gamma}{\downarrow_{\mathbb{N}}} \supset \overline{\Delta}{\downarrow_{\mathbb{N}}})$$

Definition 6.39 (valid L-sequent, minimal L-sequent)
An L-sequent $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is defined to be *valid* if its filtered sequent is valid. We denote this with $\vDash L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$. Furthermore we write $\nvDash L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ to denote that an L-sequent is not valid.

$L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is defined to be *minimal* if it is valid and if $L; \mathcal{D}'; \overline{\Gamma} \supset \overline{\Delta}$ is not valid for all proper subsets $\mathcal{D}'$ of $\mathcal{D}$.

The intention of the definition above is that L contains the labels of the consequents of a residue theory and the filter $\mathcal{D}$ defines a subset of all consequents according to their labels. The antecedent of the filtered sequent then consists of W and the subset of consequents selected by $\mathcal{D}$. We illustrate this on the previous example.

Let $\mathrm{S} := \{1,2,3\}; \mathcal{D}; \overset{0}{p_1}, \overset{0}{p_2}, \overset{1}{q_1}, \overset{2}{q_2}, \overset{3}{q_2} \supset q_1 \overset{0}{\wedge} q_2$.

$\mathcal{D} := \{1\}$**:** Then $p_1, p_2, q_1 \supset q_1 \wedge q_2$ is the filtered sequent of S. In this case S is not valid and thus also not L-minimal.

$\mathcal{D} := \{1,2\}$**:** Then $p_1, p_2, q_1, q_2 \supset q_1 \wedge q_2$ is the filtered sequent of S. In this case S is valid and also minimal.

$\mathcal{D} := \{1,2,3\}$**:** Then $p_1, p_2, q_1, q_2, q_2 \supset q_1 \wedge q_2$ is the filtered sequent of S. In this case S is valid but not minimal.

The following two lemmas indicates how two L-sequents relate according to their filtered sequent.

Lemma 6.40 (subsequent relation of filtered sequents)
Let $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ and $L'; \mathcal{D}'; \overline{\Gamma}' \supset \overline{\Delta}'$ be two L-sequents, $\Gamma \supset \Delta$ and $\Gamma' \supset \Delta'$ their filtered sequents.

If $L \supseteq L'$, $\mathcal{D} \subseteq \mathcal{D}'$, $\overline{\Gamma} \subseteq \overline{\Gamma}'$ and $\overline{\Delta} \subseteq \overline{\Delta}$ then $\Gamma \supset \Delta$ is a subsequent of $\Gamma' \supset \Delta'$.

Proof. The claim follows directly by definition. □

Lemma 6.41 (equality of filtered sequents)
Let $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ be a L-sequent, and $\overset{\mathrm{n}}{\Gamma}$ and $\overset{\mathrm{n}}{\Delta}$ be two multisets of formulas carrying label n.

1. The filtered sequents of $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ and $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ are equal iff $\mathrm{n} \in L$ or $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$.

2. The filtered sequents of $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ and $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ are equal iff $\mathrm{n} \notin \mathcal{D}$.

Proof. The two claims follow by definition. □

The following lemma discusses minimal L-sequents. It consists of two parts. The first part states that an L-sequent remains minimal if labels that do not appear in its formulas are added to or removed from its partitioner. The second part states that an L-sequent remains minimal if we add or remove formulas carrying labels that are in its partitioner but not in its filter.

Lemma 6.42 (minimal L-sequents)
Let $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ be a L-sequent, and $\overset{\mathrm{n}}{\Gamma}$ and $\overset{\mathrm{n}}{\Delta}$ be two multisets of formulas carrying label n.

1. If $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ then $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal iff $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal.

2. If $\mathrm{n} \notin \mathcal{D}$ then $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal iff $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ is minimal.

Proof.
1. $\Rightarrow$: Suppose that $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (1) and that $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal (2). From (1) we know by Lemma 6.41.1 that the filtered sequents of $L; \mathcal{D}'; \overline{\Gamma} \supset \overline{\Delta}$ and $L, \mathrm{n}; \mathcal{D}'; \overline{\Gamma} \supset \overline{\Delta}$ are equal for any $\mathcal{D}'$. With (2) we thus know that $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal.
1. $\Leftarrow$: analogously.

2. $\Rightarrow$: Suppose that $\mathrm{n} \notin \mathcal{D}$ (1) and that $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal (2). From (1) we know that $\mathrm{n} \notin \mathcal{D}'$ for $\mathcal{D}' \subseteq \mathcal{D}$. Hence we know by Lemma 6.41.2 that $L, \mathrm{n}; \mathcal{D}'; \overline{\Gamma} \supset \overline{\Delta}$ and $L, \mathrm{n}; \mathcal{D}'; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ are equal for $\mathcal{D}' \subseteq \mathcal{D}$. With (2) we thus know that $L, \mathrm{n}; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ is minimal.
2. $\Leftarrow$: analogously. □

To calculate minimal L-sequents we use a calculus that is especially tailored for that purpose.

Definition 6.43 (the calculus CPC4)
We define the classical sequent calculus CPC4 to have the deduction rules as given in Figure 6.7.

Remark 6.44 (filter vs. use-set)
In CPC4 the meaning of the filter $\mathcal{D}$ is different to the meaning of the use-set $\mathcal{D}$ in the calculus CPC2. If $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}$ is valid and $\mathrm{n} \notin \mathcal{D}$ then this does in general not imply that $\overset{\mathrm{n}}{A}$ is not needed to prove the sequent and that hence $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{B}$ is valid. The sequent $0; 0; \overset{0}{p} \supset \overset{1}{p}$ for example is valid

$$\frac{}{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{\mathbf{p}}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{p}}}(\mathsf{id})^{(a,b)} \qquad \frac{}{L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{\top}}(\top)^{(a)} \qquad \frac{}{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{\bot}\supset\overline{\Delta}}(\bot)^{(a)}$$

$$\frac{L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A}}{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{\neg\mathbf{A}}\supset\overline{\Delta}}(\neg\supset) \qquad \frac{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A}\supset\overline{\Delta}}{L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{\neg\mathbf{A}}}(\supset\neg)$$

$$\frac{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B}\supset\overline{\Delta}}{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{\mathbf{A}\wedge\mathbf{B}}\supset\overline{\Delta}}(\wedge\supset) \qquad \frac{L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B}}{L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{\mathbf{A}\vee\mathbf{B}}}(\supset\vee)$$

$$\frac{L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1;\overline{\Gamma},\overset{\mathrm{n}}{A}\supset\overline{\Delta} \quad L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{n}}{B}\supset\overline{\Delta}}{L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\vee\mathbf{B}}\supset\overline{\Delta}}(\vee\supset)$$

$$\frac{L_{\bar{\mathrm{n}}\bar{\mathrm{m}}};\mathcal{D}_1;\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A} \quad L_{\bar{\mathrm{n}}\bar{\mathrm{m}}};\mathcal{D}_2;\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{B}}{L_{\bar{\mathrm{n}}\bar{\mathrm{m}}};\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{A}\wedge\mathbf{B}}}(\supset\wedge)$$

$$\frac{L,\mathrm{m},\mathrm{n};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma},\overset{\mathrm{n}}{A}\supset\overline{\Delta}}{L,\mathrm{m};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\vee\mathbf{B}}\supset\overline{\Delta}}(\mathrm{L};\cdot\vee\supset)^{(c)} \qquad \frac{L,\mathrm{m},\mathrm{n};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma},\overset{\mathrm{n}}{B}\supset\overline{\Delta}}{L,\mathrm{m};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\vee\mathbf{B}}\supset\overline{\Delta}}(\mathrm{L};\vee\cdot\supset)^{(c)}$$

$$\frac{L,\mathrm{m},\mathrm{n};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{B}}{L,\mathrm{m};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{A}\wedge\mathbf{B}}}(\mathrm{L};\supset\cdot\wedge)^{(c)} \qquad \frac{L,\mathrm{m},\mathrm{n};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{B}}{L,\mathrm{m};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{A}\wedge\mathbf{B}}}(\mathrm{L};\supset\wedge\cdot)^{(c)}$$

$$\frac{L,\mathrm{m},\mathrm{n};\mathcal{D}_1,\mathrm{n};\overline{\Gamma},\overset{\mathrm{n}}{A}\supset\overline{\Delta} \quad L,\mathrm{m},\mathrm{n};\mathcal{D}_2,\mathrm{n};\overline{\Gamma},\overset{\mathrm{n}}{B}\supset\overline{\Delta}}{L,\mathrm{m};\mathcal{D}_1,\mathcal{D}_2,\mathrm{m};\overline{\Gamma},\overset{\mathrm{m}}{\mathbf{A}\vee\mathbf{B}}\supset\overline{\Delta}}(\mathrm{L};\vee\supset)^{(c)}$$

$$\frac{L,\mathrm{m},\mathrm{n};\mathcal{D}_1,\mathrm{n};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{A} \quad L,\mathrm{m},\mathrm{n};\mathcal{D}_2,\mathrm{n};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{n}}{B}}{L,\mathrm{m};\mathcal{D}_1,\mathcal{D}_2,\mathrm{m};\overline{\Gamma}\supset\overline{\Delta},\overset{\mathrm{m}}{\mathbf{A}\wedge\mathbf{B}}}(\mathrm{L};\supset\wedge)^{(c)}$$

$\mathrm{m},\mathrm{n}\notin L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$
$\mathrm{m},\mathrm{n}\notin\mathcal{D}_{\bar{\mathrm{n}}}$
[a] $\mathrm{n}\notin L$ or $\mathrm{n}\in\mathcal{D}$
[b] $\mathrm{m}\notin L$ or $\mathrm{m}\in\mathcal{D}$
[c] $\mathrm{n}\notin\mathrm{labels}(\overline{\Gamma}\cup\overline{\Delta})$

Figure 6.7: Deduction rules of CPC4

but $0;0;\overset{0}{p} \supset \overset{1}{q}$ is not.
However, it can easily be verified that the implication holds if $\mathrm{n} \in L$, i.e. for formulas carrying a label $\mathrm{n} \in L$ the calculus does use-check.

Theorem 6.45 (soundness of CPC4)
If an L-sequent $L;\mathcal{D};\overline{\Gamma} \supset \overline{\Delta}$ is deducible in CPC4 then it is valid.

Proof. Since the rules can be grouped into rules of similar form, we show soundness only for some selected rules.

- $\dfrac{}{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{p} \supset \overline{\Delta},\overset{\mathrm{m}}{p}}(\mathsf{id})$, where $\mathrm{n} \in L$ or $\mathrm{n} \in \mathcal{D}$, and $\mathrm{m} \in L$ or $\mathrm{m} \in \mathcal{D}$.
  Let $\Pi \supset \Sigma$ be the filtered sequent of the conclusion. Since $\mathrm{n} \in L$ or $\mathrm{n} \in \mathcal{D}$, and $\mathrm{m} \in L$ or $\mathrm{m} \in \mathcal{D}$ we know by definition that $p \in \Pi$ and $p \in \Sigma$, hence $\Pi \supset \Sigma$ is valid and thus $L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{p} \supset \overline{\Delta},\overset{\mathrm{m}}{p}$ is valid, too.

- $\dfrac{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B} \supset \overline{\Delta}}{L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A \wedge B} \supset \overline{\Delta}}(\wedge\supset)$.
  Suppose that $L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A},\overset{\mathrm{n}}{B} \supset \overline{\Delta}$ is valid and let $\Pi \supset \Sigma$ be its filtered sequent. We distinguish two cases.

  1. $\mathrm{n} \notin \mathcal{D}$ and $\mathrm{n} \in L$:
     Then $\Pi \supset \Sigma$ is also the filtered sequent of the conclusion, hence it is valid.

  2. $\mathrm{n} \in \mathcal{D}$ or $\mathrm{n} \notin L$:
     Then $\Pi \supset \Sigma$ is of the form $\Pi', A, B \supset \Sigma$. Thus $\Pi', A \wedge B \supset \Sigma$ is valid, too. It is easy to see that this is the filtered sequent of $L;\mathcal{D};\overline{\Gamma},\overset{\mathrm{n}}{A \wedge B} \supset \overline{\Delta}$. Hence the conclusion is valid.

- $\dfrac{L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1;\overline{\Gamma},\overset{\mathrm{n}}{A} \supset \overline{\Delta} \quad L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{n}}{B} \supset \overline{\Delta}}{L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{m}}{A \vee B} \supset \overline{\Delta}}(\vee\supset)$, $\mathrm{n},\mathrm{m} \notin L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$.
  Let $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1;\overline{\Gamma},\overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{n}}{B} \supset \overline{\Delta}$ be valid. Then according to Lemma 6.40 $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{n}}{B} \supset \overline{\Delta}$ are valid. Let $\Pi, A \supset \Sigma$ and $\Pi, B \supset \Sigma$ be their filtered, valid sequents. Then $\Pi, A \vee B \supset \Sigma$ is valid, too. It is easy to see that this is the filtered sequent of $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}};\mathcal{D}_1,\mathcal{D}_2;\overline{\Gamma},\overset{\mathrm{m}}{A \vee B} \supset \overline{\Delta}$. Hence the conclusion is valid.

- $\dfrac{L,\mathrm{m},\mathrm{n};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma},\overset{\mathrm{n}}{A} \supset \overline{\Delta}}{L,\mathrm{m};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma},\overset{\mathrm{m}}{A \vee B} \supset \overline{\Delta}}(\mathrm{L};\vee\supset)$, $\mathrm{n} \notin \mathcal{D}_{\bar{\mathrm{n}}}$ and $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$.
  Let $L,\mathrm{m},\mathrm{n};\mathcal{D}_{\bar{\mathrm{n}}};\overline{\Gamma},\overset{\mathrm{n}}{A} \supset \overline{\Delta}$ be valid, $\Pi \supset \Sigma$ be its filtered sequent (1), $\mathrm{n} \notin \mathcal{D}_{\bar{\mathrm{n}}}$ (2) and $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (3).
  From (1) and (2) we know by Lemma 6.41.2 that $\Pi \supset \Sigma$ is also the

filtered sequent of $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\overline{\mathrm{n}}}; \overline{\Gamma} \supset \overline{\Delta}$. From this and (3) we obtain with Lemma 6.41.1 that $\Pi \supset \Sigma$ is also the filtered sequent of $L, \mathrm{m}; \mathcal{D}_{\overline{\mathrm{n}}}; \overline{\Gamma} \supset \overline{\Delta}$. From Lemma 6.40 we know that $\Pi \supset \Sigma$ is a subsequent of the filtered sequent of $L, \mathrm{m}; \mathcal{D}_{\overline{\mathrm{n}}}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$. Hence the conclusion is valid.

- $\dfrac{L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta} \quad L, \mathrm{m}, \mathrm{n}; \mathcal{D}_2, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}}{L, \mathrm{m}; \mathcal{D}_1, \mathcal{D}_2, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}}(\mathrm{L};\vee\supset)$, $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$.

  Let $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_2, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ be valid and $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (1).

  Then we know from Lemma 6.40 that $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1, \mathcal{D}_2, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1, \mathcal{D}_2, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ are valid, too. Let $\Pi, A \supset \Sigma$ and $\Pi, B \supset \Sigma$ be their filtered, valid sequents. Then $\Pi, A \vee B \supset \Sigma$ is valid, too. Because of (1) we know that this is the filtered sequent of $L, \mathrm{m}; \mathcal{D}_1, \mathcal{D}_2, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$. Hence the conclusion is valid.

Soundness for the other rules is similar to prove. □

To show completeness of CPC4 we first introduce three lemmas. The first states that four of the rules are semantically invertible, the second shows a general relation between the validity of the premises and conclusion of the branching rules and the last one considers validity in the context of relabeled formulas.

Lemma 6.46 (semantically invertible rules of CPC4)
The CPC4-rules $(\neg \supset)$, $(\supset \neg)$, $(\wedge \supset)$ and $(\supset \vee)$ are semantically invertible.

Proof. We show our claim only for rule $(\neg \supset)$. For the other rules the claim can be proved analogously.

Suppose that $L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{\neg A} \supset \overline{\Delta}$ is valid. We distinguish two cases.

$\mathrm{n} \in L$ and $\mathrm{n} \notin \mathcal{D}$: Then the filtered sequents of the premise and the conclusion are equal, hence the premise is valid, too.

$\mathrm{n} \notin L$, or $\mathrm{n} \in L$ and $\mathrm{n} \in \mathcal{D}$: Then let $\Pi, \neg A \supset \Sigma$ be the filtered, valid sequent of the conclusion. Hence $\Pi \supset \Sigma, A$ is valid, too. Since this is the filtered sequent of the premise, we thus know that the premise is valid. □

Remark 6.47 (not semantically invertible rules of CPC4)
The rules of CPC4 that conclude a disjunction in the antecedent or a conjunction in the succedent are not semantically invertible.

For rules with two premises the reason for this is mainly to find in the arbitrary partitioning of the filter of the conclusion. Consider for example

the L-sequent $L; \mathcal{D}_1, \mathcal{D}_2; p \overset{0}{\vee} q \supset \overset{2}{p}, \overset{1}{q}$ with $L := \{1, 2\}$, $\mathcal{D}_1 := \{1\}$ and $\mathcal{D}_2 := \{2\}$. That L-sequent is valid but neither $L; \mathcal{D}_1; \overset{0}{p} \supset \overset{2}{p}, \overset{1}{q}$ nor $L; \mathcal{D}_2; \overset{0}{q} \supset \overset{2}{p}, \overset{1}{q}$ is valid.

For rules with one premise the reason for this is mainly to find in the relabeling of the active formula in the premise. Consider the L-sequent $L, 1; \mathcal{D}_{\bar{1}}; p \overset{1}{\vee} q \supset p \overset{1}{\vee} q$ with $L := \emptyset$ and $\mathcal{D}_{\bar{1}} := \{1\}$. That L-sequent is valid but $L, 1, 2; \mathcal{D}_{\bar{1}}; \overset{2}{p} \supset p \overset{1}{\vee} q$ for example is not.

If we label the active formulas in the premises with the same label as the principal formula and if we do not partition the filter of the conclusion, then the premises of a branching rule are valid if and only if the conclusion is valid.

Lemma 6.48 (validity in branching rules)
Let $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ be an L-sequent. Then

1. $\vDash L; \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{n}}{\vee} B \supset \overline{\Delta}$ iff $\vDash L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $\vDash L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$.
2. $\vDash L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, A \overset{\mathrm{n}}{\wedge} B$ iff $\vDash L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}$ and $\vDash L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{B}$.

Proof. We only show the first claim and of this only one direction. The other claim and direction can be proved analogously.

Let $L; \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{n}}{\vee} B \supset \overline{\Delta}$ be valid and $\Pi \supset \Sigma$ be its filtered, valid sequent. We distinguish two cases.

$\mathrm{n} \in L$ and $\mathrm{n} \notin \mathcal{D}$: Then $\Pi \supset \Sigma$ is also the filtered sequent of $L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ which are hence valid.

$\mathrm{n} \notin L$, or $\mathrm{n} \in L$ and $\mathrm{n} \in \mathcal{D}$: Then $\Pi \supset \Sigma$ is of the form $\Pi', A \vee B \supset \Sigma$. Thus $\Pi', A \supset \Sigma$ and $\Pi', B \supset \Sigma$ are valid. It is easy to see that they are the filtered sequents of $L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ which are hence valid. □

If we change the label of a formula in an L-sequent then we have to extend the partitioner and the filter in order to be sure that a valid L-sequent remains valid.

Lemma 6.49 (validity in the context of relabeled formulas)
Let $\overline{\Gamma}$ and $\overline{\Delta}$ be two multisets of labeled formulas, Γ and Δ be two multisets of formulas, $\mathrm{n}, \mathrm{m} \in \mathbb{N}$, $\overset{\mathrm{n}}{\Gamma} := \left\{ \overset{\mathrm{n}}{A} : A \in \Gamma \right\}$, $\overset{\mathrm{n}}{\Delta} := \left\{ \overset{\mathrm{n}}{A} : A \in \Delta \right\}$, $\overset{\mathrm{m}}{\Gamma} := \left\{ \overset{\mathrm{m}}{A} : A \in \Gamma \right\}$ and $\overset{\mathrm{m}}{\Delta} := \left\{ \overset{\mathrm{m}}{A} : A \in \Delta \right\}$.

If $\vDash L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ then $\vDash L, \mathrm{m}; \mathcal{D}, \mathrm{m}; \overline{\Gamma}, \overset{\mathrm{m}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{\Delta}$.

Proof. The claim follows directly from the fact that the filtered sequent of $L; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ is a subsequent of the one of $L, \mathrm{m}; \mathcal{D}, \mathrm{m}; \overline{\Gamma}, \overset{\mathrm{m}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{\Delta}$. □

The other direction of this lemma does not hold.
Let $\overline{\Gamma} := \overline{\Delta} := \emptyset$ and $\Gamma := \Delta := \{p\}$. Then $\vDash \mathrm{n}, \mathrm{m}; \mathrm{m}; \overset{\mathrm{m}}{p} \supset \overset{\mathrm{m}}{p}$ but $\nvDash \mathrm{n}; \emptyset; \overset{\mathrm{n}}{p} \supset \overset{\mathrm{n}}{p}$.

Theorem 6.50 (completeness of CPC4)
If an L-sequent $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is valid then it is derivable in CPC4.

Proof. Let $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ be valid and $\Pi \supset \Sigma$ be its filtered, valid sequent. We show our claim by induction on $\mathrm{len}(L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta})$.

- $\mathrm{len}(L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}) = |\overline{\Gamma}| + |\overline{\Delta}|$.

  Then $\overline{\Gamma}$ and $\overline{\Delta}$ contain only atomic formulas, hence $\Pi \supset \Sigma$ is an axiom. It is therefore either of the form $\Pi', p \supset \Sigma', p$ or $\Pi', \bot \supset \Sigma$ or $\Pi \supset \Sigma', \top$.

  Let $\Pi \supset \Sigma$ be of the form $\Pi', p \supset \Sigma', p$. Then there exist

  1. $\overset{\mathrm{n}}{p} \in \overline{\Gamma}$ such that $\mathrm{n} \notin L$, or $\mathrm{n} \in L$ and $\mathrm{n} \in \mathcal{D}$,
  2. $\overset{\mathrm{m}}{p} \in \overline{\Delta}$ such that $\mathrm{m} \notin L$, or $\mathrm{m} \in L$ and $\mathrm{m} \in \mathcal{D}$.

  Hence $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ is derivable by (id).

  The other cases are similar to prove.

- $\mathrm{len}(L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}) > |\overline{\Gamma}| + |\overline{\Delta}|$

  Then there exists a non-atomic formula $\overset{\mathrm{n}}{A}$ in $\overline{\Gamma} \cup \overline{\Delta}$. We distinguish on the form and position of $\overset{\mathrm{n}}{A}$. Since the different cases are similar to prove, we only show two cases.

  – $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta} = L; \mathcal{D}; \overline{\Gamma}', \neg \overset{\mathrm{n}}{B} \supset \overline{\Delta}$.

    Then we know by Lemma 6.46 that $L; \mathcal{D}; \overline{\Gamma}' \supset \overline{\Delta}, \overset{\mathrm{n}}{B}$ is valid and by induction hypothesis thus derivable in CPC4. With $(\neg \supset)$ we can derive $L; \mathcal{D}; \overline{\Gamma}', \neg \overset{\mathrm{n}}{B} \supset \overline{\Delta}$.

  – $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta} = L; \mathcal{D}; \overline{\Gamma}', B \overset{\mathrm{n}}{\vee} C \supset \overline{\Delta}$.

    Then we know that $L; \mathcal{D}; \overline{\Gamma}', \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ and $L; \mathcal{D}; \overline{\Gamma}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}$ are valid (Lemma 6.48). We distinguish two cases.

    * $\mathrm{n} \notin L$: By induction hypothesis we know that $L; \mathcal{D}; \overline{\Gamma}', \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ and $L; \mathcal{D}; \overline{\Gamma}', \overset{\mathrm{n}}{C} \supset \overline{\Delta}$ are derivable in CPC4. With $(\vee \supset)$ we can derive $L; \mathcal{D}; \overline{\Gamma}', B \overset{\mathrm{n}}{\vee} C \supset \overline{\Delta}$.

∗ $\mathrm{n} \in L$: Then let $\mathrm{m} \in \mathbb{N}$ such that $\mathrm{m} \notin \mathrm{labels}(\overline{\Gamma}' \cup \overline{\Delta})$ (1). We distinguish two cases.

· $\mathrm{n} \in \mathcal{D}$: Then we know that $L, \mathrm{m}; \mathcal{D}, \mathrm{m}; \overline{\Gamma}', \overset{\mathrm{m}}{B} \supset \overline{\Delta}$ and $L, \mathrm{m}; \mathcal{D}, \mathrm{m}; \overline{\Gamma}', \overset{\mathrm{m}}{C} \supset \overline{\Delta}$ are valid (Lemma 6.49). By induction hypothesis they are thus derivable in CPC4. With $(L; \vee \supset)$ we can derive $L; \mathcal{D}; \overline{\Gamma}', B \overset{\mathrm{n}}{\vee} C \supset \overline{\Delta}$.

· $\mathrm{n} \notin \mathcal{D}$: Then let $\Pi \supset \Sigma$ be the filtered sequent of $L; \mathcal{D}; \overline{\Gamma}', B \overset{\mathrm{n}}{\vee} C \supset \overline{\Delta}$.
Because $\mathrm{n} \in L$ we know by Lemma 6.41.2 that $\Pi \supset \Sigma$ is also the filtered sequent of $L; \mathcal{D}; \overline{\Gamma}' \supset \overline{\Delta}$.
From this and (1) we know by Lemma 6.41.1 that $\Pi \supset \Sigma$ is also the filtered sequent of $L, \mathrm{m}; \mathcal{D}; \overline{\Gamma}' \supset \overline{\Delta}$.
Because of (1) we know by definition that $\mathrm{m} \notin \mathcal{D}$. We thus know by Lemma 6.41.2 that $\Pi \supset \Sigma$ is also the filtered sequent of $L, \mathrm{m}; \mathcal{D}; \overline{\Gamma}', \overset{\mathrm{m}}{B} \supset \overline{\Delta}$.
Hence $L, \mathrm{m}; \mathcal{D}; \overline{\Gamma}', \overset{\mathrm{m}}{B} \supset \overline{\Delta}$ is valid. By induction hypothesis we know that it is derivable in CPC4. With $(L; \cdot \vee \supset)$ we can derive $L; \mathcal{D}; \overline{\Gamma}', B \overset{\mathrm{n}}{\vee} C \supset \overline{\Delta}$.

□

Minimality in CPC4

The idea we now follow is to use backward proof search in CPC4 to calculate all minimal L-sequents. We thereby rely on two properties.

- In non-branching rules the conclusion is minimal if and only if the premise is minimal.
- In branching rules minimal conclusion can always be obtained from two minimal premises.

These two properties allow us to concentrate only on minimal sequents during proof search in order to calculate all minimal sequents.
We first show that for non-branching rules the minimal sets of the premise is equal to those of its conclusion.

The CPC4-rules $(\neg \supset)$, $(\supset \neg)$, $(\wedge \supset)$ and $(\supset \vee)$ respect minimality. This means that if the premise of a rule is minimal then the conclusion is also minimal and vice versa.

Theorem 6.51 (minimality in $(\neg \supset)$, $(\supset \neg)$, $(\wedge \supset)$ and $(\supset \vee)$)
Let $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ be an L-sequent. Then

1. $L;\mathcal{D};\overline{\Gamma},\overset{\text{n}}{A}\supset\overline{\Delta}$ is minimal iff $L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{\neg A}$ is minimal.
2. $L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{A}$ is minimal iff $L;\mathcal{D};\overline{\Gamma},\overset{\text{n}}{\neg A}\supset\overline{\Delta}$ is minimal.
3. $L;\mathcal{D};\overline{\Gamma},A\overset{\text{n}}{\wedge}B\supset\overline{\Delta}$ is minimal iff $L;\mathcal{D};\overline{\Gamma},\overset{\text{n}}{A},\overset{\text{n}}{B}\supset\overline{\Delta}$ is minimal.
4. $L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},A\overset{\text{n}}{\vee}B$ is minimal iff $L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{A},\overset{\text{n}}{B}$ is minimal.

Proof. Since we can prove all claims analogously we only show claim 1.

$\Rightarrow$: Let $L;\mathcal{D};\overline{\Gamma},\overset{\text{n}}{A}\supset\overline{\Delta}$ be minimal.
Then $\vDash L;\mathcal{D};\overline{\Gamma},\overset{\text{n}}{A}\supset\overline{\Delta}$ (1) and $\nvDash L;\mathcal{D}';\overline{\Gamma},\overset{\text{n}}{A}\supset\overline{\Delta}$ for all $\mathcal{D}'\subsetneq\mathcal{D}$ (2).
From (1) we know by soundness of $\mathsf{CPC4}$ that $\vDash L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{\neg A}$. From (2) we know by contraposition of Lemma 6.46 that $\nvDash L;\mathcal{D}';\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{\neg A}$ for all $\mathcal{D}'\subsetneq\mathcal{D}$. Hence $L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{\neg A}$ is minimal.

$\Leftarrow$: Let $L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{\neg A}$ be minimal.
Then $\vDash L;\mathcal{D};\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{\neg A}$ (1) and $\nvDash L;\mathcal{D}';\overline{\Gamma}\supset\overline{\Delta},\overset{\text{n}}{\neg A}$ for all $\mathcal{D}'\subsetneq\mathcal{D}$ (2).
From (1) we know by Lemma 6.46 that $\vDash L;\mathcal{D};\overline{\Gamma},\overset{\text{n}}{A}\supset\overline{\Delta}$. From (2) we know by soundness of $\mathsf{CPC4}$ that $\nvDash L;\mathcal{D}';\overline{\Gamma},\overset{\text{n}}{A}\supset\overline{\Delta}$ for all $\mathcal{D}'\subsetneq\mathcal{D}$. Hence $L;\mathcal{D};\overline{\Gamma},\overset{\text{n}}{A}\supset\overline{\Delta}$ is minimal. □

Those rules of $\mathsf{CPC4}$ that deduce a disjunction in the antecedent or a conjunction in the succedent do not respect minimality.

Consider for example the rule $(L;\cdot\vee\supset)$. Applying it on a minimal premise does in the following example not result in a minimal conclusion.

$$\frac{\{1,2,3\};\{1,2\};\overset{1}{p},\overset{3}{p}\supset p\overset{2}{\vee}q}{\{1,2\};\{1,2\};\overset{1}{p},p\overset{2}{\vee}q\supset p\overset{2}{\vee}q}(\text{L};\cdot\vee\supset)$$

Regarding rule $(L;\vee\ \supset)$ consider the following example, in which a non-minimal conclusion is deduced from two minimal premises.

$$\frac{\{1,2,3,4\};\{1,4\};\overset{4}{p}\supset p\overset{1}{\vee}q,q\overset{2}{\vee}r\quad\{1,2,3,4\};\{2,4\};\overset{4}{q}\supset p\overset{1}{\vee}q,q\overset{2}{\vee}r}{\{1,2,3\};\{1,2,3\};p\overset{3}{\vee}q\supset p\overset{1}{\vee}q,q\overset{2}{\vee}r}(\text{L};\vee\supset)$$

A similar example can be given for rule $(\vee\supset)$.

$$\frac{\{1,2\};\{1\};\overset{4}{p}\supset p\overset{1}{\vee}q,q\overset{2}{\vee}r\quad\{1,2\};\{2\};\overset{4}{q}\supset p\overset{1}{\vee}q,q\overset{2}{\vee}r}{\{1,2\};\{1,2\};p\overset{3}{\vee}q\supset p\overset{1}{\vee}q,q\overset{2}{\vee}r}(\vee\supset)$$

However, we can show that for these rules a minimal conclusion can always be computed from minimal premises. We start with the rules $(\vee\ \supset)$ and $(\supset\ \wedge)$.

Proposition 6.52 (minimality in CPC4 (i))
Let $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}$ be an L-sequent, $\mathrm{m} \notin L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$ and $\mathrm{n} \notin L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$.

1. If $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ is minimal then there exist $\mathcal{D}_1$ and $\mathcal{D}_2$ such that $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}_1; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}_2; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ are minimal and $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$.

2. If $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, A \overset{\mathrm{m}}{\wedge} B$ is minimal then there exist $\mathcal{D}_1$ and $\mathcal{D}_2$ such that $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}_1; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{A}$ and $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}_2; \overline{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{B}$ are minimal and $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$.

Proof. Both claims are similar to proof. We only show the first one.
Suppose that $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ is minimal (1), $\mathrm{m} \notin L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$ and $\mathrm{n} \notin L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$. Then $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ are valid. Let $\mathcal{D}_1, \mathcal{D}_2 \subseteq \mathcal{D}$ be such that $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}_1; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}_2; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ are minimal. Then $L_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \mathcal{D}_1, \mathcal{D}_2; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ is valid. Furthermore $\mathcal{D}_1 \cup \mathcal{D}_2 \subseteq \mathcal{D}$. From (1) we know $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$. □

We now show for the rules $(L; \vee\ \supset)$, $(L; \cdot\vee\ \supset)$, $(L; \vee\cdot\ \supset)$, $(L; \supset\ \wedge)$, $(L; \supset\ \cdot\wedge)$ and $(L; \supset\ \wedge\cdot)$ that a minimal conclusion can always be derived from minimal premises. To do so we distinguish between the label of the principal formula being in the filter of the conclusion or not.

The case where the label of the principal formula is in the filter of the conclusion is covered by the following proposition.

Proposition 6.53 (minimality in CPC4 (ii))
Let $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}; \overline{\Gamma} \supset \overline{\Delta}$ be an L-sequent, $\mathrm{m} \notin \mathcal{D}_{\bar{\mathrm{m}}}$ and $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$.

1. If $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ is minimal then it can be derived in CPC4 from minimal premises.

2. If $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma} \supset \overline{\Delta}, A \overset{\mathrm{m}}{\wedge} B$ is minimal then it can be derived in CPC4 from minimal premises.

Proof. Both claims are similar to prove. We only show the first claim.
Let $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ be minimal (1), $\mathrm{m} \notin \mathcal{D}_{\bar{\mathrm{m}}}$ (2), $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (3). We distinguish three cases.

1. $\mathrm{m} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (a).
Then because of (2) we know that the filtered sequents of $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}; \overline{\Gamma} \supset \overline{\Delta}$

and $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ are equal. From (1) we know that the latter L-sequent is not valid, hence $\nvDash L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}; \overline{\Gamma} \supset \overline{\Delta}$ (b).

From (1) we know $\vDash L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$, hence

$\vDash L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $\vDash L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$.

With (a) we obtain

$\vDash L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ (c) and $\vDash L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ (d).

Let $\mathcal{D}_1, \mathcal{D}_2 \subseteq \mathcal{D}_{\bar{\mathrm{m}}} \cup \{\mathrm{n}\}$ be such that

$L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_2; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ are minimal (e).

We claim that $\mathrm{n} \in \mathcal{D}_1$ and $\mathrm{n} \in \mathcal{D}_2$ and show our claim for the first case. Suppose that $\mathrm{n} \notin \mathcal{D}_1$ (f). Then we know that the filtered sequent of $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ is equal to the one of $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1; \overline{\Gamma} \supset \overline{\Delta}$. With (f) we obtain $\mathcal{D}_1 \subseteq \mathcal{D}_{\bar{\mathrm{m}}}$. With (b) we thus know that those two L-sequents are not valid. This conflicts to (e), hence $\mathrm{n} \in \mathcal{D}_1$.

So $\mathcal{D}_1$ and $\mathcal{D}_2$ are of the form $\mathcal{D}_{1\bar{\mathrm{n}}} \cup \{\mathrm{n}\}$ and $\mathcal{D}_{2\bar{\mathrm{n}}} \cup \{\mathrm{n}\}$, with $\mathrm{n} \notin \mathcal{D}_{1\bar{\mathrm{n}}}$ and $\mathrm{n} \notin \mathcal{D}_{2\bar{\mathrm{n}}}$. We hence can use rule $(L; \vee \supset)$ on them:

$$\frac{L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{1\bar{\mathrm{n}}}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta} \quad L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{2\bar{\mathrm{n}}}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}}{L, \mathrm{m}; \mathcal{D}_{1\bar{\mathrm{n}}}, \mathcal{D}_{2\bar{\mathrm{n}}}, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}}(\mathrm{L};\vee\supset)$$

Now $\mathcal{D}_{1\bar{\mathrm{n}}}, \mathcal{D}_{2\bar{\mathrm{n}}} \subseteq \mathcal{D}_{\bar{\mathrm{m}}}$, hence $\mathcal{D}_{1\bar{\mathrm{n}}} \cup \mathcal{D}_{2\bar{\mathrm{n}}} \subseteq \mathcal{D}_{\bar{\mathrm{m}}}$. With (1) we know $\mathcal{D}_{1\bar{\mathrm{n}}} \cup \mathcal{D}_{2\bar{\mathrm{n}}} = \mathcal{D}_{\bar{\mathrm{m}}}$.

2. $\mathrm{m} \in \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (a) and $\nvDash L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma} \supset \overline{\Delta}$ (b).
From (1) we know

$\vDash L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $\vDash L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$.

Let $\mathcal{D}_1, \mathcal{D}_2 \subseteq \mathcal{D}_{\bar{\mathrm{m}}} \cup \{\mathrm{n}, \mathrm{m}\}$ be such that

$L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ and $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_2; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}$ are minimal (c).

We claim that $\mathrm{n} \in \mathcal{D}_1$ and $\mathrm{n} \in \mathcal{D}_2$ and show our claim for the first case. Suppose that $\mathrm{n} \notin \mathcal{D}_1$ (d).

Then we know that the filtered sequent of $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ is equal to the one of $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_1; \overline{\Gamma} \supset \overline{\Delta}$. From (d) we obtain $\mathcal{D}_1 \subseteq \mathcal{D}_{\bar{\mathrm{m}}} \cup \{\mathrm{m}\}$. With (b) we thus know that those two L-sequents are not valid. This conflicts to (c), hence $\mathrm{n} \in \mathcal{D}_1$.

So $\mathcal{D}_1$ and $\mathcal{D}_2$ are of the form $\mathcal{D}_{1\bar{\mathrm{n}}} \cup \{\mathrm{n}\}$ and $\mathcal{D}_{2\bar{\mathrm{n}}} \cup \{\mathrm{n}\}$, with $\mathrm{n} \notin \mathcal{D}_{1\bar{\mathrm{n}}}$ and $\mathrm{n} \notin \mathcal{D}_{2\bar{\mathrm{n}}}$. We hence can use rule $(L; \vee \supset)$ on them:

$$\frac{L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{1\bar{\mathrm{n}}}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta} \quad L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{2\bar{\mathrm{n}}}, \mathrm{n}; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta}}{L, \mathrm{m}; \mathcal{D}_{1\bar{\mathrm{n}}}, \mathcal{D}_{2\bar{\mathrm{n}}}, \mathrm{m}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}}(\mathrm{L};\vee\supset)$$

Now $\mathcal{D}_{1\bar{\mathrm{n}}}, \mathcal{D}_{2\bar{\mathrm{n}}} \subseteq \mathcal{D}_{\bar{\mathrm{m}}} \cup \{\mathrm{m}\}$, hence $\mathcal{D}_{1\bar{\mathrm{n}}} \cup \mathcal{D}_{2\bar{\mathrm{n}}} \cup \{\mathrm{m}\} \subseteq \mathcal{D}_{\bar{\mathrm{m}}} \cup \{\mathrm{m}\}$. With (1) we obtain $\mathcal{D}_{1\bar{\mathrm{n}}} \cup \mathcal{D}_{2\bar{\mathrm{n}}} \cup \{\mathrm{m}\} = \mathcal{D}_{\bar{\mathrm{m}}} \cup \{\mathrm{m}\}$.

3. $\mathrm{m} \in \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (a) and $\vDash L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma} \supset \overline{\Delta}$ (b).
With (1) and (b) it is easy to see that $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal (c).
From (2) and (a) we know that $\mathrm{n} \neq \mathrm{m}$ (d).
From (3) we know by definition that $\mathrm{n} \notin \mathcal{D}_{\bar{\mathrm{m}}}$ (e).
With Lemma 6.42.1 we know from (c) and (3) that $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}, \mathrm{n}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal.
From (d) and (e) we then know by Lemma 6.42.2 that $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ is minimal. With $(L; \cdot\vee\ \supset)$ we can derive $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}}, \mathrm{m}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ from it. □

The case where the label of the principal formula is not in the filter of the conclusion is covered by the following proposition.

Proposition 6.54 (minimality in CPC4 (iii))
Let $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \overline{\Gamma} \supset \overline{\Delta}$ be an L-sequent, $\overset{\mathrm{n}}{\Gamma}, \overset{\mathrm{n}}{\Delta}$ and $\overset{\mathrm{m}}{\Gamma}, \overset{\mathrm{m}}{\Delta}$ be multisets of formulas carrying label n and m, respectively, $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ and $\mathrm{n}, \mathrm{m} \notin \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$.

Then $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ is minimal iff $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \overline{\Gamma}, \overset{\mathrm{m}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{\Delta}$ is minimal.

Proof. Let $\mathrm{m}, \mathrm{n} \notin \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}$ (1) and $\mathrm{n} \notin \mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})$ (2).
From (1) we know by Lemma 6.42.2 that $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \overline{\Gamma}, \overset{\mathrm{n}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{n}}{\Delta}$ is minimal iff $L, \mathrm{m}, \mathrm{n}; \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \overline{\Gamma} \supset \overline{\Delta}$ is minimal. From (2) and Lemma 6.42.1 we know that this is equivalent to $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \overline{\Gamma} \supset \overline{\Delta}$ being minimal. From (1) and Lemma 6.42.2 we know that this is equivalent to $L, \mathrm{m}; \mathcal{D}_{\bar{\mathrm{m}}\bar{\mathrm{n}}}; \overline{\Gamma}, \overset{\mathrm{m}}{\Gamma} \supset \overline{\Delta}, \overset{\mathrm{m}}{\Delta}$ being minimal. □

The three propositions above lead to the following theorem. It states that for those rules that do not respect minimality, a minimal conclusion can be deduced from minimal premises.

Theorem 6.55 (minimality in CPC4))
Let $\overline{\Gamma}$ and $\overline{\Delta}$ be two multisets of formulas.

1. If $L; \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ is minimal then it can be derived in CPC4 from minimal premises.

2. If $L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta}, A \overset{\mathrm{m}}{\wedge} B$ is minimal then it can be derived in CPC4 from minimal premises.

Proof. Both claims are similar to prove. We only show claim 1.
Let $L;\mathcal{D};\overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ be minimal. Then we distinguish three cases.

$\mathrm{m} \notin L$: Then the claim follows from proposition 6.52.

$\mathrm{m} \notin \mathcal{D}$: Then we know from Proposition 6.54 that $L, \mathrm{m}, \mathrm{n}; \mathcal{D}; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta}$ is minimal. With $(L; \cdot\vee\supset)$ we can derive $L, \mathrm{m}; \mathcal{D}; \overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ from it.

$\mathrm{m} \in \mathcal{D}$: Then the claim follows from proposition 6.53. □

From the theorem above we know how to recursively calculate minimal sequents for rules that do not respect minimality. We show this on an example where we deduce a disjunction in the antecedent. Let

$$\mathbb{D} := \left\{ \mathcal{D} : L;\mathcal{D};\overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta} \text{ is minimal} \right\}$$

and n be a fresh label. We then consider two cases.

1. $\mathrm{m} \notin L$: This case reflects rule $(\vee\supset)$. Let

$$\mathbb{D}_1 := \left\{ \mathcal{D}_1 : L;\mathcal{D}_1;\overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta} \text{ is minimal} \right\},$$
$$\mathbb{D}_2 := \left\{ \mathcal{D}_2 : L;\mathcal{D}_2;\overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta} \text{ is minimal} \right\},$$
$$\mathbb{D}_{12} := \{ \mathcal{D}_1 \cup \mathcal{D}_2 : \mathcal{D}_1 \in \mathbb{D}_1, \mathcal{D}_2 \in \mathbb{D}_2 \}.$$

   $\mathcal{D}_1$ and $\mathcal{D}_2$ contain those filters that make the left and right premise minimal. According to the theorem above we know that $\mathbb{D} \subseteq \mathbb{D}_{12}$. Furthermore we know from the correctness of CPC4 that $L;\mathcal{D};\overline{\Gamma}, A \overset{\mathrm{m}}{\vee} B \supset \overline{\Delta}$ is valid for $\mathcal{D} \in \mathbb{D}_{12}$. Hence

$$\mathbb{D} = \{ \mathcal{D} \in \mathbb{D}_{12} : \mathcal{D} \text{ is minimal regarding } \subseteq \text{ in } \mathbb{D}_{12} \}.$$

2. $\mathrm{m} \in L$: This case reflects rule $(L;\vee\supset)$, $(L;\cdot\vee\supset)$ or $(L;\vee\cdot\supset)$. Let

$$\mathbb{D}_1 := \left\{ \mathcal{D}_1 : L, \mathrm{n}; \mathcal{D}_1; \overline{\Gamma}, \overset{\mathrm{n}}{A} \supset \overline{\Delta} \text{ is minimal} \right\},$$
$$\mathbb{D}_2 := \left\{ \mathcal{D}_2 : L, \mathrm{n}; \mathcal{D}_2; \overline{\Gamma}, \overset{\mathrm{n}}{B} \supset \overline{\Delta} \text{ is minimal} \right\},$$
$$\mathbb{D}_{1\mathrm{n}} := \{ \mathcal{D} \in \mathbb{D}_2 : \mathrm{n} \in \mathcal{D} \}, \quad \mathbb{D}_{2\mathrm{n}} := \{ \mathcal{D} \in \mathbb{D}_1 : \mathrm{n} \in \mathcal{D} \},$$
$$\mathbb{D}_{1\bar{\mathrm{n}}} := \{ \mathcal{D} \in \mathbb{D}_1 : \mathrm{n} \notin \mathcal{D} \}, \quad \mathbb{D}_{2\bar{\mathrm{n}}} := \{ \mathcal{D} \in \mathbb{D}_2 : \mathrm{n} \notin \mathcal{D} \}.$$
$$\mathbb{D}_{12} := \{ ((\mathcal{D}_1 \cup \mathcal{D}_2) \setminus \{\mathrm{n}\}) \cup \{\mathrm{m}\} : \mathcal{D}_1 \in \mathbb{D}_{1\mathrm{n}}, \mathcal{D}_2 \in \mathbb{D}_{2\mathrm{n}} \} \cup \mathbb{D}_{1\bar{\mathrm{n}}}.$$

   $\mathcal{D}_{1\mathrm{n}}$ and $\mathcal{D}_{2\mathrm{n}}$ contain those filters that make the left and right premise of rule $(L;\vee\supset)$ minimal. $\mathcal{D}_{1\bar{\mathrm{n}}}$ and $\mathcal{D}_{2\bar{\mathrm{n}}}$ contain those filters that make

the premise of rule $(L;\cdot\vee\ \supset)$ or $(L;\vee\cdot\ \supset)$ minimal. According to Lemma 6.42.2 we know that $\mathcal{D}_{1\bar{\mathrm{n}}} = \mathcal{D}_{2\bar{\mathrm{n}}}$.

Now $\mathcal{D}_{12}$ contains those filters that derive from a deduction on minimal premises. Its first part reflects rule $(L;\vee\ \supset)$, its second part reflects the other two rules.
With the same argumentation as in case 1 we obtain

$$\mathbb{D} = \{\mathcal{D} \in \mathbb{D}_{12} : \mathcal{D} \text{ is minimal regarding } \subseteq \text{ in } \mathbb{D}_{12}\}.$$

Minimal Axioms

We now know how to compute the minimal conclusions from the minimal premises. The problem that remains is to compute the minimal axioms. To find these we need a different approach than the one we use for normal proof search.

In normal proof search we return with success as soon as an axiom is found. In general we don't find all minimal axioms with this approach. Consider the following example.

$$\frac{1;1;\overset{0}{\mathbf{p}},\overset{0}{q} \supset \overset{1}{\mathbf{p}},\overset{1}{q},p \overset{0}{\wedge} q}{1;1;\overset{0}{p},\overset{0}{q} \supset \mathbf{p} \overset{1}{\vee} \mathbf{q},p \overset{0}{\wedge} q}(\supset\vee)$$

Here we encounter a non-minimal axiom. We have still a non-atomic formula in that axiom. If we process it we end up with two sequents containing only atomic formulas. For these sequents we can then choose those filters that make them minimal. In our example the empty filter is the only such filter.

$$\frac{\dfrac{1;;\overset{0}{\mathbf{p}},\overset{0}{q} \supset \overset{1}{p},\overset{1}{q},\overset{2}{\mathbf{p}} \quad 1;;\overset{0}{p},\overset{0}{\mathbf{q}} \supset \overset{1}{p},\overset{1}{q},\overset{2}{\mathbf{q}}}{1;;\overset{0}{p},\overset{0}{q} \supset \overset{1}{p},\overset{1}{q},\mathbf{p} \overset{0}{\wedge} \mathbf{q}}(\supset\wedge)}{1;;\overset{0}{p},\overset{0}{q} \supset \mathbf{p} \overset{1}{\vee} \mathbf{q},p \overset{0}{\wedge} q}(\supset\vee)$$

The general approach to find the minimal filters is thus to continue to backward apply rules until we end up in a sequent that contains only atomic formulas. We may stop processing formulas if we encounter an axiom with an empty filter, because then we know that this is the only filter that turns the sequent minimal.

An Algorithm to Compute Minimal Sequents

We now have a method to find the minimal axioms and know how to compute the minimal sequents of the conclusion of a rule from the minimal sequents

of its premises and can thus give an algorithm that computes the filters $\mathbb{D}$ that make a sequent minimal.

The pseudocode is given in Algorithm 15 and is split up into two functions whose arguments are all passed by value.

minFilters is used as interface function and to classify formulas. It takes as arguments a partitioner L and two multisets of labeled formulas representing the sequent for which we want to find the minimal filters. For lack of space we only give the pseudocode for formulas in the antecedent. The processing of formulas in the succedent is analogical.

First (lines 2–4) we partition the given sequent into the pairs $\langle \overline{\Gamma}_a, \overline{\Delta}_a \rangle$ (atomic formulas), $\langle \overline{\Gamma}_b, \overline{\Delta}_b \rangle$ (formulas that need a branching rule to be deduced) and $\langle \overline{\Gamma}_c, \overline{\Delta}_c \rangle$ (formulas that need a non-branching rule to be deduced).

Then we decide on the result of the partitioning how to proceed.

If there is an axiom with an empty filter we return the set containing the empty set (lines 5–6).
If there are formulas left to classify, we classify one of them by calling minFilters again with the premises of the backward applied rule (lines 7–12).
If there are no formulas left to classify but formulas that require a branching rule to be deduced we call branch to backward apply a branching rule (lines 13 + 14).
If only atomic formulas are left we return all filters $\mathcal{D}$ that for which the given sequent is minimal (lines 15 + 16).

branch applies a branching rules backwards. Its arguments are equal to those of minFilters.
For lack of space we only give the pseudocode for the formulas in the antecedent. The processing of the formulas in the succedent is analogical.

First we calculate a fresh label n with which to label the active formulas in the premise and calculate the formulas that need a branching rule to be deduced (lines 20 + 21).

We select a branching formula (line 22). If the label of that formula is not in L then we backward apply rule ($\vee \supset$), call minFilters for the premises and calculate the set $\mathbb{D}_{12}$ of filters that result from applying the rule onto minimal premises (lines 23–26). If the label of that formula is in L then we proceed analogously (lines 27–33). Then we calculate the set $\mathbb{D}$ of minimal filters of the current sequent (line 34).

Algorithm 15 Computing minimal filters

function minFilters(L, $\overline{\Gamma}$, $\overline{\Delta}$)
  $\overline{\Gamma}_a := \{\overset{\mathrm{n}}{A} \in \overline{\Gamma} : \overset{\mathrm{n}}{A} \text{ is atomic}\}$; $\overline{\Delta}_a := \{\overset{\mathrm{n}}{A} \in \overline{\Delta} : \overset{\mathrm{n}}{A} \text{ is atomic}\}$
  $\overline{\Gamma}_b := \{\overset{\mathrm{n}}{A} \in \overline{\Gamma} : \overset{\mathrm{n}}{A} = B \overset{\mathrm{n}}{\vee} C\}$; $\overline{\Delta}_b := \{\overset{\mathrm{n}}{A} \in \overline{\Delta} : \overset{\mathrm{n}}{A} = B \overset{\mathrm{n}}{\wedge} C\}$
  $\overline{\Gamma}_c := \overline{\Gamma} \setminus (\overline{\Gamma}_a \cup \overline{\Gamma}_b)$; $\overline{\Delta}_c := \overline{\Delta} \setminus (\overline{\Delta}_a \cup \overline{\Delta}_b)$
  if $L; \emptyset; \overline{\Gamma}_a \supset \overline{\Delta}_a$ is an axiom **then**
    return $\{\emptyset\}$
  if $\overline{\Gamma}_c \neq \emptyset$ **then** choose $\overset{\mathrm{n}}{A} \in \overline{\Gamma}_c$
    if $\overset{\mathrm{n}}{A} = \overset{\mathrm{n}}{\neg} B$ **then**
      $\mathbb{D} :=$ minFilters(L, $\overline{\Gamma} \setminus \{\overset{\mathrm{n}}{A}\}$, $\overline{\Delta} \cup \{\overset{\mathrm{n}}{B}\}$)
    else ▷ $\overset{\mathrm{n}}{A} = B \overset{\mathrm{n}}{\wedge} C$
      $\mathbb{D} :=$ minFilters(L, $(\overline{\Gamma} \setminus \{\overset{\mathrm{n}}{A}\}) \cup \{\overset{\mathrm{n}}{B}, \overset{\mathrm{n}}{C}\}$, $\overline{\Delta}$)
  else if $\overline{\Delta}_c \neq \emptyset$ **then** proceed analogously with $\overline{\Delta}_c$
  else if $\overline{\Gamma}_b \cup \overline{\Delta}_b \neq \emptyset$ **then**
    $\mathbb{D} :=$ branch(L, $\overline{\Gamma}$, $\overline{\Delta}$)
  else ▷ only atoms left
    $\mathbb{D} := \{\mathcal{D} : L; \mathcal{D}; \overline{\Gamma} \supset \overline{\Delta} \text{ is minimal}\}$
  return $\mathbb{D}$

function branch(L, $\overline{\Gamma}$, $\overline{\Delta}$)
  $\mathrm{n} := \max(\mathrm{labels}(\overline{\Gamma} \cup \overline{\Delta})) + 1$
  $\overline{\Gamma}_b := \{\overset{\mathrm{m}}{A} \in \overline{\Gamma} : \overset{\mathrm{m}}{A} = B \overset{\mathrm{m}}{\vee} C\}$; $\overline{\Delta}_b := \{\overset{\mathrm{m}}{A} \in \overline{\Delta} : \overset{\mathrm{m}}{A} = B \overset{\mathrm{m}}{\wedge} C\}$
  if $\overline{\Gamma}_b \neq \emptyset$ **then** choose $\overset{\mathrm{m}}{A} \in \overline{\Gamma}_b$ ▷ $\overset{\mathrm{m}}{A} = B \overset{\mathrm{m}}{\vee} C$
    if $\mathrm{m} \notin L$ **then**
      $\mathbb{D}_1 :=$ minFilters(L, $(\overline{\Gamma} \setminus \{\overset{\mathrm{m}}{A}\}) \cup \{\overset{\mathrm{n}}{B}\}$, $\overline{\Delta}$)
      $\mathbb{D}_2 :=$ minFilters(L, $(\overline{\Gamma} \setminus \{\overset{\mathrm{m}}{A}\}) \cup \{\overset{\mathrm{n}}{C}\}$, $\overline{\Delta}$)
      $\mathbb{D}_{12} := \{\mathcal{D}_1 \cup \mathcal{D}_2 : \mathcal{D}_1 \in \mathbb{D}_1, \mathcal{D}_2 \in \mathbb{D}_2\}$
    else
      $\mathbb{D}_1 :=$ minFilters($L \cup \{\mathrm{n}\}$, $(\overline{\Gamma} \setminus \{\overset{\mathrm{m}}{A}\}) \cup \{\overset{\mathrm{n}}{B}\}$, $\overline{\Delta}$)
      $\mathbb{D}_2 :=$ minFilters($L \cup \{\mathrm{n}\}$, $(\overline{\Gamma} \setminus \{\overset{\mathrm{m}}{A}\}) \cup \{\overset{\mathrm{n}}{C}\}$, $\overline{\Delta}$)
      $\mathbb{D}_{1\bar{n}} := \{\mathcal{D} \in \mathbb{D}_1 : \mathrm{m} \notin \mathcal{D}\}$
      $\mathbb{D}_{1n} := \mathbb{D}_1 \setminus \mathbb{D}_{1\bar{n}}$
      $\mathbb{D}_{2n} := \mathbb{D}_2 \setminus \mathbb{D}_{1\bar{n}}$
      $\mathbb{D}_{12} := \{((\mathcal{D}_1 \cup \mathcal{D}_2) \setminus \{\mathrm{n}\}) \cup \{\mathrm{m}\} : \mathcal{D}_1 \in \mathbb{D}_{1n} \text{ and } \mathcal{D}_2 \in \mathbb{D}_{2n}\} \cup$
        $\mathbb{D}_{1\bar{n}}$
    $\mathbb{D} := \{\mathcal{D} \in \mathbb{D}_{12} : \mathcal{D} \text{ is minimal in } \mathbb{D}_{12} \text{ regarding set inclusion}\}$
  else proceed analogously with $\overline{\Delta}_b$
  return $\mathbb{D}$

A Simple Example

We illustrate our algorithm on the sequent $1,2,3;\mathcal{D};\overset{1}{p},\neg p \overset{1}{\vee} r,\neg p \overset{2}{\vee} r \supset \overset{3}{r}$. We call minFilters$(\{1,2,3\},\left\{\overset{1}{p},\neg p \overset{1}{\vee} r,\neg p \overset{2}{\vee} r\right\},\left\{\overset{3}{r}\right\})$ to calculate the filters that make the sequent minimal.
Since the given sequent is not an axiom and all its non-atomic formulas require a branching rule to be deduced we call branch with the given arguments (line 14).

There we calculate a fresh label n = 4 (line 20), choose the formula $\neg p \overset{1}{\vee} r$ from $\overline{\Gamma}_b$ and call minFilters to calculate $\mathbb{D}_1$ (line 28).

- minFilters$(\{1,2,3,4\},\left\{\overset{1}{p},\overset{4}{\neg p},\neg p \overset{2}{\vee} r\right\},\left\{\overset{3}{r}\right\})$

  The given sequent is no axiom but has the formula $\overset{4}{\neg p}$ that needs to be classified. We thus apply $(\neg\supset)$ (line 9) and call
  minFilters$(\{1,2,3,4\},\left\{\overset{1}{p},\neg p \overset{2}{\vee} r\right\},\left\{\overset{3}{r},\overset{4}{p}\right\})$
  Now we encounter an axiom. But since we still have non-atomic formulas in our sequent and $1,2,3,4;\emptyset;\overset{1}{p},\neg p \overset{2}{\vee} r \supset \overset{3}{r},\overset{4}{p}$ is not valid, we continue and call branch with the given arguments (line 14).
  There we calculate a fresh label n = 5 (line 20), choose the formula $\neg p \overset{2}{\vee} r$ from $\overline{\Gamma}_b$ and call minFilters to calculate $\mathbb{D}_1$ (line 28).

  - minFilters$(\{1,2,3,4,5\},\left\{\overset{1}{p},\overset{5}{\neg p}\right\},\left\{\overset{3}{r},\overset{4}{p}\right\})$

    Since $1,2,3,4,5;\emptyset;\overset{1}{p},\overset{5}{\neg p} \supset \overset{3}{r},\overset{4}{p}$ is not valid and the formula $\overset{5}{\neg p}$ is left to classify we apply $(\neg\supset)$ (line 9) and call
    minFilters$(\{1,2,3,4,5\},\left\{\overset{1}{p}\right\},\left\{\overset{3}{r},\overset{4}{p},\overset{5}{p}\right\})$
    Now there are only atomic formulas left. We calculate the minimal filters $\mathbb{D}$ of the given sequent and return them (lines 16–17).

  We obtain $\mathbb{D}_1 = \{\{1,4\},\{1,5\}\}$ from minFilters (line 28) and call it again to calculate $\mathbb{D}_2$ (line 29).

  - minFilters$(\{1,2,3,4,5\},\left\{\overset{1}{p},\overset{5}{r}\right\},\left\{\overset{3}{r},\overset{4}{p}\right\})$
    Since there are only atomic formulas in this sequent we calculate the minimal filters $\mathbb{D}$ and return them.

  We obtain $\mathbb{D}_2 = \{\{1,4\},\{3,5\}\}$ from minFilters (line 29).
  Now we calculate $\mathbb{D}_{12}$ (lines 30–33) for $n = 5$ and $m = 2$. $\mathbb{D}_{1n} = \{\{1,5\}\}$, $\mathbb{D}_{1\bar{n}} = \{\{1,4\}\}$, $\mathbb{D}_{2n} = \{\{3,5\}\}$, $\mathbb{D}_{12} = \{\{1,2,3\},\{1,4\}\}$
  Then we calculate the minimal sets $\mathbb{D} = \{\{1,2,3\},\{1,4\}\}$ (line 34) and return them.

The two sets of $\mathbb{D}$ correspond to the following three deductions where we write L_{1n} to denote the set $\{1, 2, \ldots, n\}$.

$$\dfrac{\dfrac{\dfrac{}{L_{15}; 1, 4; \overset{1}{\mathbf{p}} \supset \overset{3}{r}, \overset{4}{\mathbf{p}}, \overset{5}{p}}(\mathsf{id})}{L_{15}; 1, 4; \overset{1}{p}, \neg\overset{4}{\mathbf{p}} \supset \overset{3}{r}, \overset{5}{p}}(\neg\supset)}{L_{14}; 1, 4; \overset{1}{p}, \neg\overset{2}{\mathbf{p}} \vee \mathbf{r} \supset \overset{3}{r}, \overset{4}{p}}(\mathrm{L};\cdot\vee\supset) \qquad \dfrac{\dfrac{}{L_{15}; 1, 4; \overset{1}{\mathbf{p}}, \overset{5}{r} \supset \overset{3}{r}, \overset{4}{\mathbf{p}}}(\mathsf{id})}{L_{14}; 1, 4; \overset{1}{p}, \neg\overset{2}{\mathbf{p}} \vee \mathbf{r} \supset \overset{3}{r}, \overset{4}{p}}(\mathrm{L};\vee\cdot\supset)}$$

$$\dfrac{\dfrac{\dfrac{}{L_{15}; 1, 5; \overset{1}{\mathbf{p}} \supset \overset{3}{r}, \overset{4}{p}, \overset{5}{\mathbf{p}}}(\mathsf{id})}{L_{15}; 1, 5; \overset{1}{p}, \neg\overset{5}{\mathbf{p}} \supset \overset{3}{r}, \overset{4}{p}}(\neg\supset) \qquad \dfrac{}{L_{15}; 3, 5; \overset{1}{p}, \overset{5}{\mathbf{r}} \supset \overset{3}{\mathbf{r}}, \overset{4}{p}}(\mathsf{id})}{L_{14}; 1, 2, 3; \overset{1}{p}, \neg\overset{2}{\mathbf{p}} \vee \mathbf{r} \supset \overset{3}{r}, \overset{4}{p}}(\mathrm{L};\vee\supset)$$

We obtain $\mathbb{D}_1 = \{\{1, 2, 3\}, \{1, 4\}\}$ from minFilters and call it again to calculate $\mathbb{D}_2$ (line 29).

- minFilters$(\{1, 2, 3, 4\}, \left\{\overset{1}{p}, \overset{4}{r}, \neg\overset{2}{p} \vee r\right\}, \left\{\overset{3}{r}\right\})$
  Since there are non-atomic formulas in the given sequent and because $1, 2, 3, 4; \emptyset; \overset{1}{p}, \overset{4}{r}, \neg\overset{2}{p} \vee r \supset \overset{3}{r}$ is not valid we continue and call branch with the given arguments (line 14). There we calculate a fresh label $n = 5$ (line 20), choose the formula $\neg\overset{2}{p} \vee r$ from $\overline{\Gamma}_b$ and call minFilters to calculate $\mathbb{D}_1$ (line 28).
  - minFilters$(\{1, 2, 3, 4, 5\}, \left\{\overset{1}{p}, \overset{4}{r}, \neg\overset{5}{p}\right\}, \left\{\overset{3}{r}\right\})$
    Since $1, 2, 3, 4, 5; \emptyset; \overset{1}{p}, \overset{4}{r}, \neg\overset{5}{p} \supset \overset{3}{r}$ is not valid and the formula $\neg\overset{5}{p}$ is left to classify we apply $(\neg\supset)$ (line 9) and call minFilters$(\{1, 2, 3, 4, 5\}, \left\{\overset{1}{p}, \overset{4}{r}\right\}, \left\{\overset{3}{r}, \overset{5}{p}\right\})$
    Now there are only atomic formulas left. We calculate the minimal filters $\mathbb{D}$ of the given sequent and return them (lines 16–17).

  We obtain $\mathbb{D}_1 = \{\{1, 5\}, \{3, 4\}\}$ from minFilters (line 28) and call it again to calculate $\mathbb{D}_2$ (line 29).
  - minFilters$(\{1, 2, 3, 4, 5\}, \left\{\overset{1}{p}, \overset{4}{r}, \overset{5}{r}\right\}, \left\{\overset{3}{r}\right\})$
    Since there are only atomic formulas in this sequent we calculate the minimal filters $\mathbb{D}$ and return them.

  We obtain $\mathbb{D}_2 = \{\{3, 4\}, \{3, 5\}\}$ from minFilters (line 29).
  Now we calculate $\mathbb{D}_{12}$ (lines 30–33) for $n = 5$ and $m = 2$. $\mathbb{D}_{1n} = \{\{1, 5\}\}$, $\mathbb{D}_{1\bar{n}} = \{\{3, 4\}\}$, $\mathbb{D}_{2n} = \{\{3, 5\}\}$, $\mathbb{D}_{12} = \{\{1, 2, 3\}, \{3, 4\}\}$
  Then we calculate the minimal sets $\mathbb{D} = \{\{1, 2, 3\}, \{3, 4\}\}$ (line 34) and return them.

The two sets of $\mathbb{D}$ correspond to the following three deductions where we write L_{1n} to denote the set $\{1, 2, \ldots, n\}$.

$$\dfrac{\dfrac{\dfrac{}{L_{15}; 3, 4; \overset{1}{p}, \overset{4}{\mathbf{r}} \supset \overset{3}{\mathbf{r}}, \overset{5}{p}}(\mathsf{id})}{L_{15}; 3, 4; \overset{1}{p}, \overset{4}{r}, \overset{5}{\neg\mathbf{p}} \supset \overset{3}{r}}(\neg\supset)}{L_{14}; 3, 4; \overset{1}{p}, \overset{4}{r}, \neg\overset{2}{\mathbf{p}} \vee \mathbf{r} \supset \overset{3}{r}}(\mathrm{L};\cdot\vee\supset) \qquad \dfrac{\dfrac{}{L_{15}; 3, 4; \overset{1}{p}, \overset{4}{\mathbf{r}}, \overset{5}{r} \supset \overset{3}{\mathbf{r}}}(\mathsf{id})}{L_{14}; 3, 4; \overset{1}{p}, \overset{4}{r}, \neg\overset{2}{\mathbf{p}} \vee \mathbf{r} \supset \overset{3}{r}}(\mathrm{L};\vee\cdot\supset)}$$

$$\dfrac{\dfrac{\dfrac{}{L_{15}; 1, 5; \overset{1}{\mathbf{p}}, \overset{4}{r} \supset \overset{3}{r}, \overset{5}{\mathbf{p}}}(\mathsf{id})}{L_{15}; 1, 5; \overset{1}{p}, \overset{4}{r}, \overset{5}{\neg\mathbf{p}} \supset \overset{3}{r}}(\mathsf{id}) \qquad \dfrac{}{L_{15}; 3, 5; \overset{1}{p}, \overset{4}{r}, \overset{5}{\mathbf{r}} \supset \overset{3}{\mathbf{r}}}(\mathsf{id})}{L_{14}; 1, 2, 3; \overset{1}{p}, \overset{4}{r}, \neg\overset{2}{\mathbf{p}} \vee \mathbf{r} \supset \overset{3}{r}}(\mathrm{L};\vee\supset)$$

We obtain $\mathbb{D}_2 = \{\{1, 2, 3\}, \{3, 4\}\}$ from minFilters (line 29). Now we calculate $\mathbb{D}_{12}$ (lines 30–33) for $n = 4$ and $m = 1$. $\mathbb{D}_{1n} = \{\{1, 4\}\}$, $\mathbb{D}_{1\bar{n}} = \{\{1, 2, 3\}\}$, $\mathbb{D}_{2n} = \{\{3, 4\}\}$, $\mathbb{D}_{12} = \{\{1, 3\}, \{1, 2, 3\}\}$.
Then we calculate from $\mathbb{D}_{12}$ the minimals sets $\mathbb{D} = \{\{1, 3\}\}$ and thus know that the only minimal sequent is $1, 2, 3; 1, 3; \overset{1}{p}, \neg p \overset{1}{\vee} r, \neg p \overset{2}{\vee} r \supset \overset{3}{r}$.

6.3 Experimental Results

In this section we compare the different proving approaches on some scalable problems. The problems mainly aim to point out the advantages and disadvantages of the approaches. All problems were computed under Debian Linux 4.0 on an AMD Sempron 2600+ with 1.5 GB RAM.

To prove whether a residue sequent is valid we have investigated four main approaches.

1. Backward apply the deduction rules and prefer rule (Res1) to (Res2). For this approach we have investigated the following provers. PR1, implementing no improvements (cf. Algorithm 6, p. 129). PR1g, implementing general improvement (cf. Algorithm 7, p. 135). PR1u, implementing use-check (cf. Algorithm 7, p. 142). PR1gu, implementing general improvement and use-check (cf. Algorithm 9, p. 146).

2. Backward apply of the deduction rules and prefer rule (Res2) to (Res1). For this approach we have investigated the following provers. PR2 implementing no improvements. PR2g implementing general improvement. PR2u implementing use-check. PR2gu implementing general improvement and use-check.

3. Calculate $\mathrm{Cl}'(W, R)$ and use the CPC prover to verify the validity of the sequent (cf. Algorithm 11 on page 151). When referring to this approach we speak of the prover PRcl.

4. Compute the minimal quasi-supports to calculate $\mathrm{Cl}'(W, R)$ partially and then use the CPC prover to verify the validity of the sequent (cf. Algorithm 13, p. 159). When referring to this approach we speak of the prover PRqs.

For each problem we investigate a sorted and a random scenario. That is we use a sorted and a random order in which the residues are processed. To have comparable results for the random scenario we initialize the random number generator with the number of residues in the theory before shuffling.

For most problems the prover PR1, PR1g, PR1gu, PR2 and PR2g turn out to be slow. We therefore examine those provers only for the first few problems and mention them in later problems only if they perform similar or better than the other provers.

We use graphics to visualize the given residue theories. To depict the residue A/B we use arrows: $A \longrightarrow B$. To depict that the formula A is in a theory we draw a circle around it: $\textcircled{A}$. We combine the two elements. The following graph for example depicts the theory $A, A/B, B/A$: $\textcircled{A} \longleftrightarrow B$.

6.3.1 Problem 1: Chain of Residues

We start with a very simple residue theory that is defined as follows.

$$T_1(n) := \{p_1\} \cup \left\{ \frac{p_i}{p_{i+1}} : 1 \leq i \leq n \right\}$$

In this theory we have a chain of residues and thus an easy closure of $T_1(n)$, $\mathrm{Th}(\{p_1, p_2, \ldots, p_{n+1}\})$. Hence the prerequisites of all residues are in the extension. Depicted as a graph $T_1(n)$ looks as follows.

$$\textcircled{p_1} \longrightarrow p_2 \longrightarrow \cdots \longrightarrow p_n$$

In the sorted scenario we use the index of the prerequisite as sort key.

Proving $T_1(n) \supset p_{\frac{n}{2}+1}$

The first problem is set in a way that we have n residues and need half of them to prove the sequent.

Slow Provers
PR1, PR1g, PR1gu, PR1u, PR2 and PR2g show similar bad performances (cf. Figure 6.8 and 6.9). We analyze these provers for $n = 20$ (cf. Table 6.1 and 6.2).

PR1 and PR1u perform equally. Use-check never succeeds for this problem. The analysis therefore reveals the same figures for both provers.
In the sorted scenario we backward apply each residue rule about 3 million times and the provers succeed in about 10 000 cases for (Res1) and in about 500 cases for (Res2). There are about 3 million calls of the CPC prover and about 1000 of them succeed.
In the random scenario the figures for these two provers are a little bit better. We backward apply each rule about 1 million times and the provers succeeds in about 65 000 cases for (Res1) and in 10 cases for (Res2). There are about 1 million calls of the CPC prover and about 40 000 of them succeed.

PR1g and PR1gu perform marginally better than PR1.
In the sorted scenario general improvement is successful in 45 cases. We backward apply each residue rule about 1 million times. Thereby (Res1) succeeds in 155 and (Res2) in 10 cases. There are about 1 million calls of the CPC prover and only 11 of them succeed.
In the random scenario general improvement succeeds in about 40 000 cases. We backward apply each residue rule about 1 million times. Thereby (Res1) succeeds in 104 and (Res2) in 10 cases. There are about 1 million call of the CPC prover and only 11 of them succeed.

The fact that PR1 performs so bad is not astonishing since half of the residues of the theory are needed to prove the problem and the strategy of the prover is to drop the residues before taking it into consideration. This way a lot of backtracking has to be done. Since the improvements we implemented are of little use, the poor performance of those provers is also not astonishing.

PR2 shows the worst performance. This is not astonishing since this prover branches for every residue it processes and therefore proving time increases exponentially.
In the sorted scenario we backward apply each residue rule about 5 million times. Thereby both rules succeed in about 500'000 cases. There are about 6 million calls of the CPC prover and about 1 million of them succeed.
In the random scenario the figures become even worse. We backward apply (Res1) again about 5 million and (Res2) about 6 million times and succeed in about 800 000 cases for (Res1) and about 1 million cases for (Res2). There are about 7 million calls of the CPC prover and about 2 million of them succeed.

PR2g behaves marginally better than PR2.
In the sorted scenario general improvement succeeds in half a million cases.

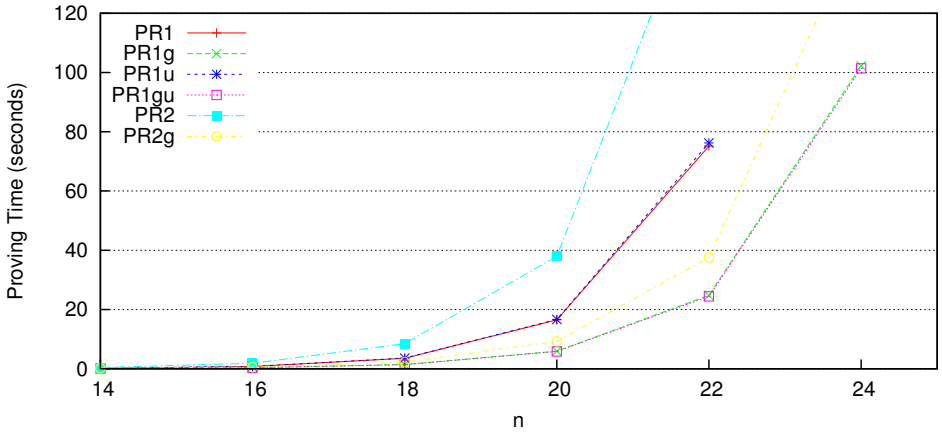


Figure 6.8: Proving time of $T_1(n) \supset p_{\frac{n}{2}+1}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR1/PR1u	2.9E6	2.9E6	–	–/0	2.9E6
PR1g/PR1gu	1.0E6	1.0E6	45	–/0	1.0E6
PR2	5.0E6	5.5E6	–	–	6.0E6
PR2g	524E3	1.0E6	524E3	–	1.0E6

Table 6.1: Figures of proving $T_1(20) \supset p_{11}$ (sorted order)

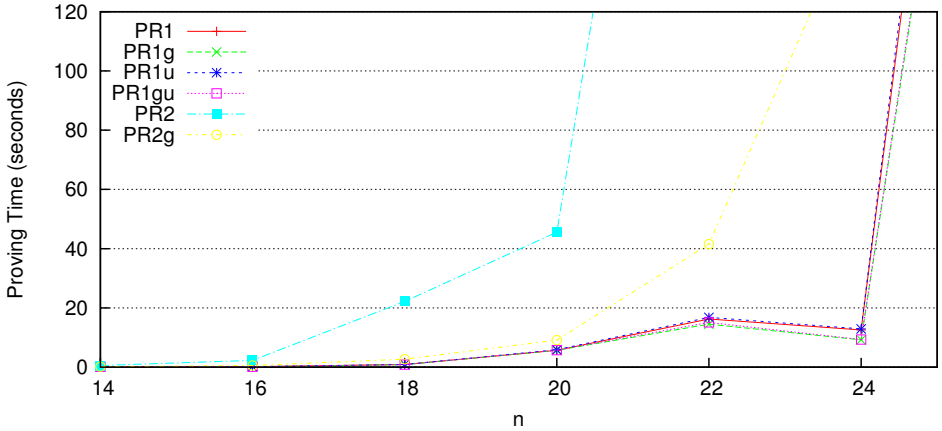


Figure 6.9: Proving time of $T_1(n) \supset p_{\frac{n}{2}+1}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR1/PR1u	1.0E6	0.9E6	–	–/0	1.0E6
PR1g/PR1gu	0.9E6	0.9E6	41E3	–/0	0.9E6
PR2	5.1E6	6.1E6	–	–	7.1E6
PR2g	525E3	1.1E6	525E3	–	1.1E6

Table 6.2: Figures of proving $T_1(20) \supset p_{11}$ (random order)

We backward apply (Res1) about half a million (Res2) about a million times and succeed in all but 171 cases for (Res1) and about half a million cases for (Res2). There are about a million calls of the CPC prover of which all but 19 succeed.
In the random scenario we have similar numbers.

Fast Provers
We have much better performance for PR2u, PR2gu, PRqs and PRcl (cf. Figure 6.10). We analyze these provers for $n = 1000$ (cf. Table 6.3 and 6.4).

PR2u and PR2gu perform similar good. The analysis reveals that use-check is responsible for the good performance. General improvement is of no use here in combination with use-check. We backward apply (Res1) never and (Res2) 375 750 times in the sorted and 247 487 times in the random scenario. In both scenarios none of those backward applications fails. Use-check is successful in all but 500 cases. There are 501 calls of the CPC prover and all of them succeed.

PRqs performs very well. Since there are only atomic formulas involved, the task of the algorithm for this problem is not much more than to find n times a minimal axiom. The analysis confirms this. In both scenarios there are 501 calls of the CPC prover and all of them succeed.
The performance of the prover drops a little bit if the residues are processed in a random order.

PRcl performs best if the residues are processed in sorted order. It performs so well because this order is optimal for that prover.
At first the residue p_1/p_2 is processed. Since p_1 is provable from $W = \{p_1\}$ we add p_2 to our theory and continue with the residue p_2/p_3. Now p_2 is provable from $\{p_1, p_2\}$ and we add p_3 to our theory. This scheme continues until all residues are processed.
The analysis confirms this. There are 1001 calls of the CPC prover and none of them fails.
The prover performs worse if the residues are processed in a random and therefore not optimal order. Now there are about 250 000 calls of the CPC prover of which only 1001 succeed.

Observation
For this problem we observe that PRcl depends heavily on the order in which it processes the residues. The provers PR2u, PR2gu and PRqs are less independent on the processing order and behave similar in both scenarios. Use-check is especially useful for the provers that prioritize the branching residue rule.

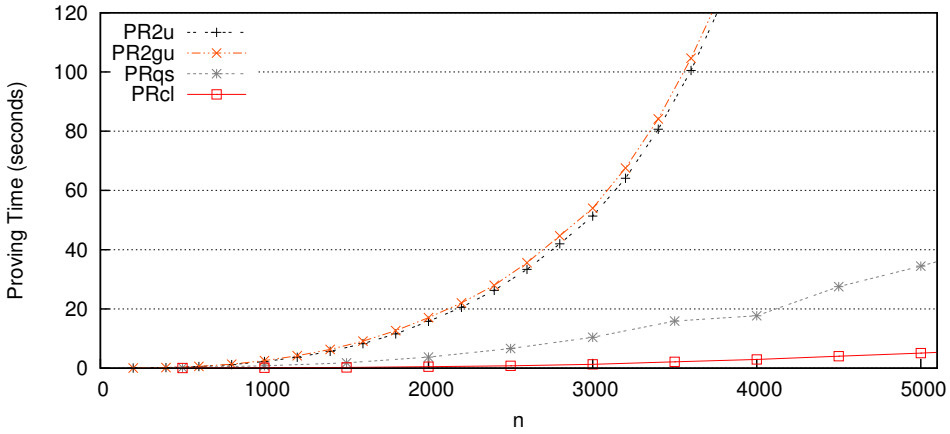


Figure 6.10: Proving time of $T_1(n) \supset p_{n+1}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	376E3	0/–	375E3	501
PRcl	–	–	–	–	1001
PRqs	–	–	–	–	501

Table 6.3: Figures of proving $T_1(1000) \supset p_{501}$ (sorted order)

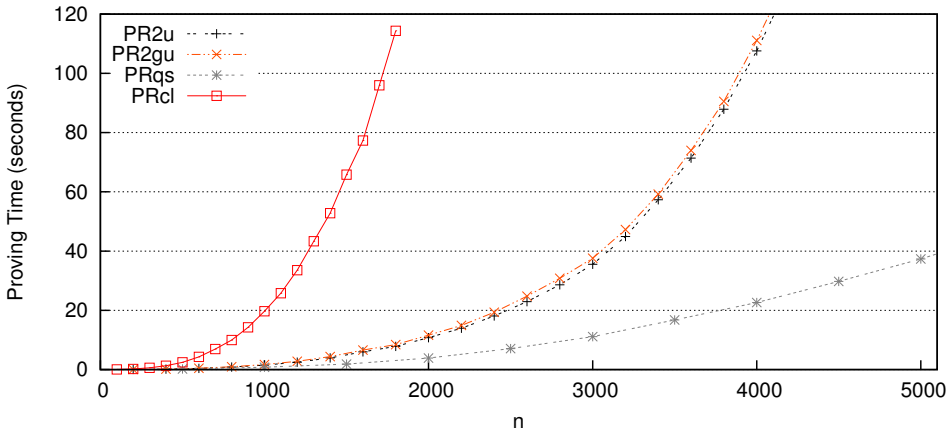


Figure 6.11: Proving time of $T_1(n) \supset p_{n+1}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	248E3	0/–	247E3	501
PRcl	–	–	–	–	256E3
PRqs	–	–	–	–	501

Table 6.4: Figures of proving $T_1(1000) \supset p_{501}$ (random order)

Proving $T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$

The second problem is set in a way that we have n residues and need at least one and at most all of them to prove the sequent.

Slow Provers
For this problem we have similar results for the slower provers (cf. Figure 6.12). PR2 is still slowest followed by PR2g. Both provers don't show much change in performance when processing the residues randomly. PR1, PR1g, PR1u, and PR1gu all perform equally. Their performance improves strongly when the residues are processed in random order. However, the improvement does not hold for all orders. In our tests there are some cases where the proving time is as bad as with a sorted processing order.

We analyze the problem for these provers for $n = 20$. For PR2 and PR2g we have exactly the same figures as for the previous problem.
The figures of the other provers for the sorted case are identical for PR1, PR1g, PR1u, and PR1gu. Each residue rule is applied about half a million time. For (Res1) the provers are successful in 209 cases, for (Res2) in just one case. Neither general improvement nor use-check is successful. The CPC prover is called half a million times and succeeds in 21 cases.
Randomizing the processing order leads in most cases to much better performance. The figures of all provers are similar. Rule (Res1) is applied about 100 and succeeds in about 40 cases. Rule (Res2) is applied 64 times and succeeds in one case. General improvement is successful in 3 cases for PR1g and PR1gu. The CPC prover is called about 70 times and succeeds in about 10 cases.

Fast Provers
Again we have much better performance for PR2u, PR2gu, PRqs and PRcl (cf. Figure 6.14 and 6.15). We analyze the provers for $n = 1000$ (cf. Table 6.7 and 6.8).

The performance of PRcl is equal to the previous problem. This was to be expected since the calculation of the closure remains the same. Compared to the other provers PRcl is again best with a sorted but worst with a random processing order of the residues.

PRqs is in the midfield of the fast provers. In contrast to the previous problem we now have n minimal quasi-supports instead of 1. Therefore the prover performs worse than for the first problem. The processing order has again only a marginal effect on its performance.

PR2u and PR2gu perform more or less equally.
In contrast to the provers that base on PR1 the figures for both scenarios are equal. This has a simple reason. After all the first premises are processed

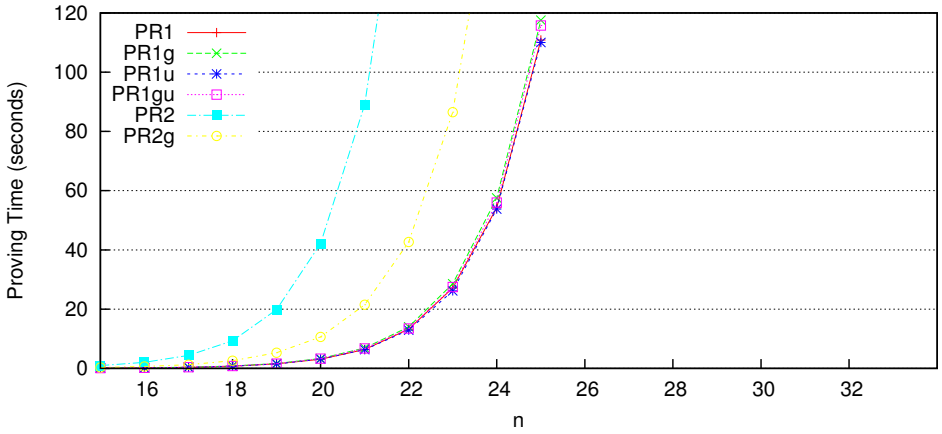


Figure 6.12: Proving time of $T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR1/PR1u	524E3	524E3	–	–/0	524E3
PR1g/PR1gu	524E3	524E3	0	–/0	524E3
PR2	5.0E6	5.5E6	–	–	6.0E6
PR2g	524E3	1.0E6	524E3	–	1.0E6

Table 6.5: Figures of proving $T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (sorted order)

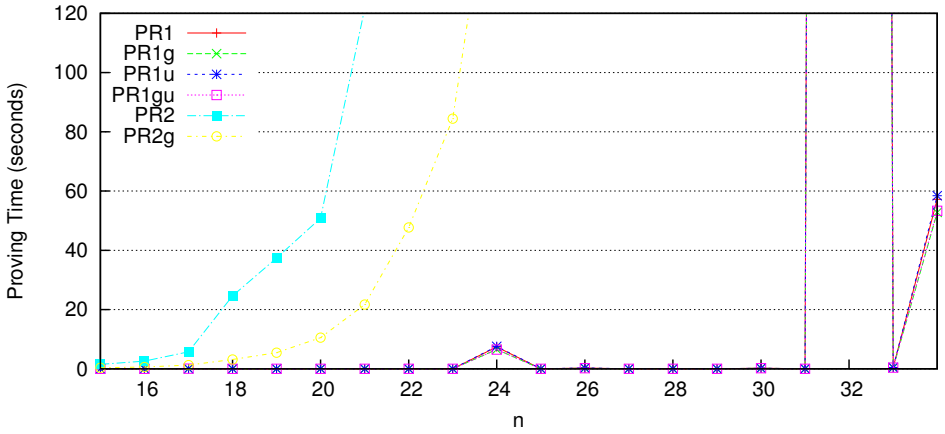


Figure 6.13: Proving time of $T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR1/PR1u	108	64	–	–/0	76
PR1g/PR1gu	105	64	3	–/0	73
PR2	5.1E6	6.1E6	–	–	7.1E6
PR2g	525E3	1.1E6	525E3	–	1.1E6

Table 6.6: Figures of proving $T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (random order)

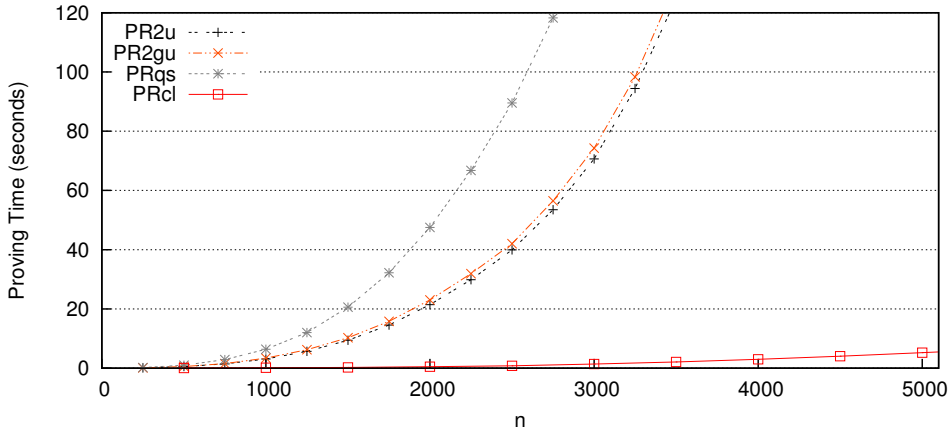


Figure 6.14: Proving time of $T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	501E3	0/–	500E3	1001
PRcl	–	–	–	–	1001
PRqs	–	–	–	–	1001

Table 6.7: Figures of proving $T_1(1000) \supset p_2 \vee \cdots \vee p_{1001}$ (sorted order)

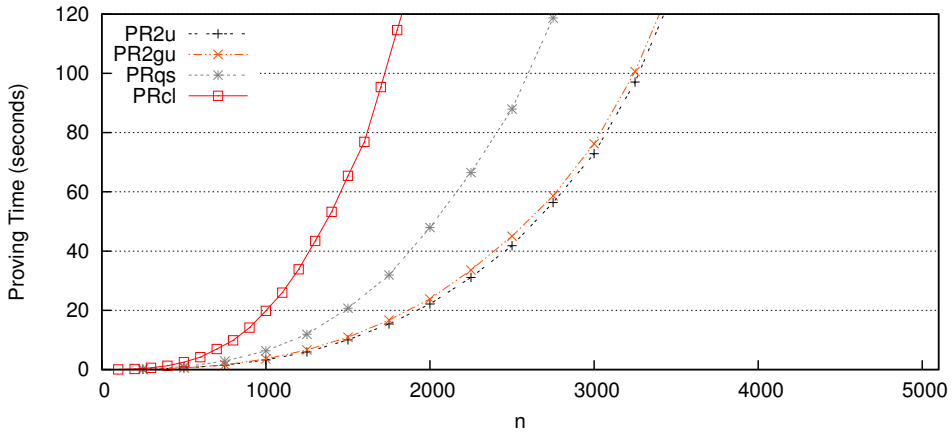


Figure 6.15: Proving time of $T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	501E3	0/–	500E3	1.0E3
PRcl	–	–	–	–	256E3
PRqs	–	–	–	–	1.0E3

Table 6.8: Figures of proving $T_1(1000) \supset p_2 \vee \cdots \vee p_{1001}$ (random order)

the prover reaches the sequent $p_1, \ldots, p_{n+1} \supset p_2 \vee \cdots \vee p_{n+1}$. It then calls the CPC prover which backward applies $(\supset \vee)$ and succeeds with the axiom $p_1, \ldots, \mathbf{p_{n+1}} \supset p_2 \vee \cdots \vee p_n, \mathbf{p_{n+1}}$. Choosing that axiom results in have the last residue in the chain marked as used. Therefore we have to show for all the residues in our theory that the prerequisite is in the closure. Because of use-check we then only branch where necessary and get the same performance for both scenarios.

Rewriting the Succedent
The picture changes if we rewrite the formula in the succedent to $p_{n+1} \vee \cdots \vee p_2$. When processing the residues in sorted order the prover now shows the best performance and a steady behavior (cf. Figure 6.16). When using a random order the prover performs equally good or better but shows an unsteady behavior (cf. Figure 6.17).
The unsteady behavior has a simple reason. After all first premises are processed the prover reaches an axiom with p_2 as principal formulas. Hence the first residue in the chain will be marked as used. Use-check will then succeed in all nodes but the one where p_1/p_2 is processed. In that node the prover has to prove the second premise. Now the closer this node is to the root node of the search the more residues have to be processed in the second branch and the longer the proof of the second branch will take. Processing p_1/p_2 in the root node — as in the sorted scenario — is thus the worst case for the chosen axiom. In the random scenario p_1/p_2 is processed at an arbitrary position, therefore proving time may vary and does not grow steadily.

Observation
For this problem we observe that PRqs drops in performance when the number of quasi-supports grows. Nevertheless, when processing the residues randomly it still performs better than PRcl.
PR2gu and PR2u perform better than PRqs. A random processing order can improve the performance of these provers if use-check is successful in most of the cases.

Proving $T_1(n) \supset p_2 \wedge \cdots \wedge p_{n+1}$

The last problem we are going to analyze for $T_1(n)$ is the conjunction of all its residues consequents. There is one minimal quasi-support and we need the consequents of all residues to prove the validity of $T_1(n) \supset p_2 \wedge \cdots \wedge p_{n+1}$.

Fast Provers
Since PR2, PR2g and the provers based on PR1 perform as bad as before, we only discuss the results of the other provers (cf. Figure 6.18 and 6.19). We analyze the provers for $n = 1000$ (cf. Table 6.11 and 6.12).

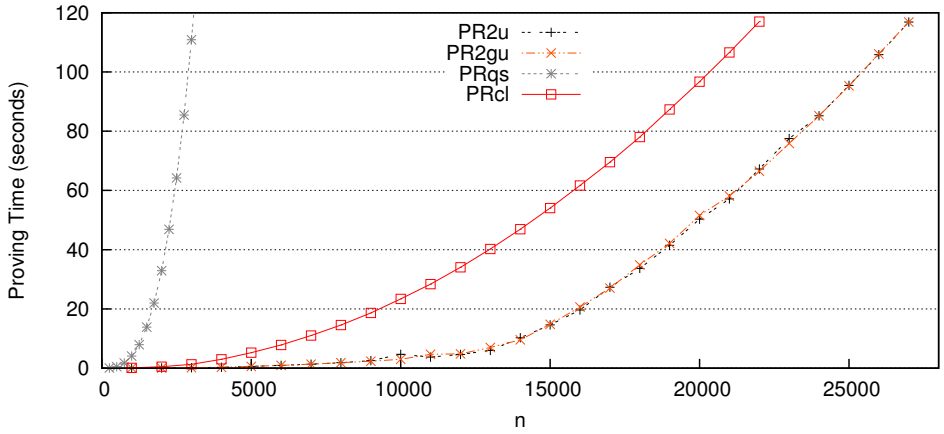


Figure 6.16: Proving time of $T_1(n) \supset p_{n+1} \vee \cdots \vee p_2$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	1999	0/–	1998	2
PRcl	–	–	–	–	1001
PRqs	–	–	–	–	2

Table 6.9: Figures of proving $T_1(1000) \supset p_{1001} \vee \cdots \vee p_2$ (sorted order)

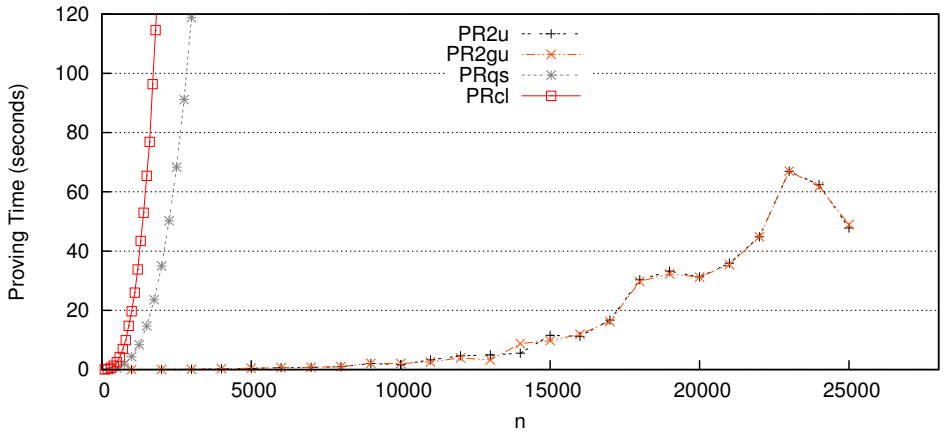


Figure 6.17: Proving time of $T_1(n) \supset p_{n+1} \vee \cdots \vee p_2$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	1386	0/–	1385	2
PRcl	–	–	–	–	256E3
PRqs	–	–	–	–	2

Table 6.10: Figures of proving $T_1(1000) \supset p_{1001} \vee \cdots \vee p_2$ (random order)

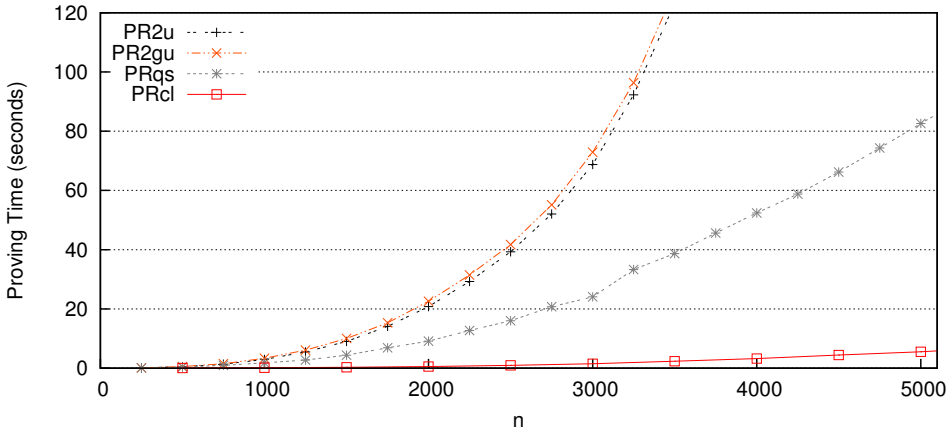


Figure 6.18: Proving time of $T_1(n) \supset p_2 \wedge \cdots \wedge p_{n+1}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	501E3	0/–	500E3	1001
PRcl	–	–	–	–	1001
PRqs	–	–	–	–	1001

Table 6.11: Figures of proving $T_1(1000) \supset p_2 \wedge \cdots \wedge p_{1001}$ (sorted order)

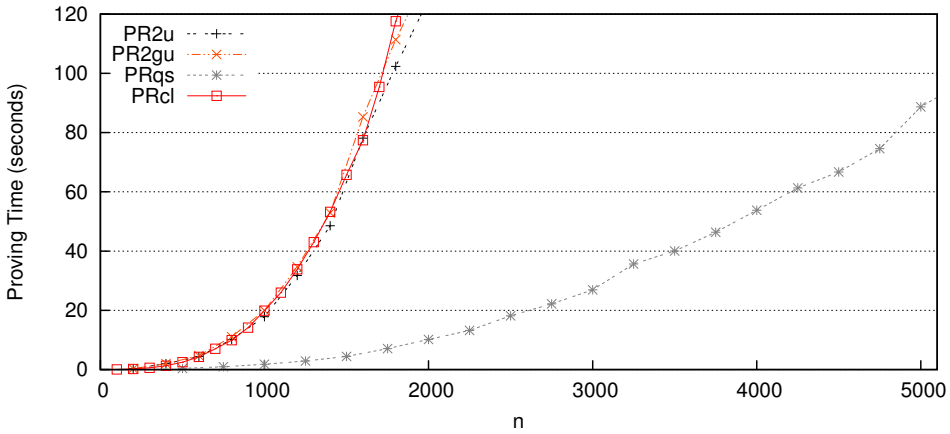


Figure 6.19: Proving time of $T_1(n) \supset p_2 \wedge \cdots \wedge p_{n+1}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	2.8E6	0/–	2.8E6	6.1E3
PRcl	–	–	–	–	256E3
PRqs	–	–	–	–	1.0E3

Table 6.12: Figures of proving $T_1(1000) \supset p_2 \wedge \cdots \wedge p_{1001}$ (random order)

PRcl shows the same performance as for the previous problems.
PRqs performs better than for the previous problem. This is not astonishing, since for this problem we only have one quasi-support. With a random processing order the performance is even a bit better.
PR2u and PR2gu perform in the sorted scenario equally well as when proving $p_2 \vee \cdots \vee p_{n+1}$. However, for the random scenario the proving times are doubled and the provers show the same performance as PRcl. The figures confirm this observation. For $n = 1000$ there are nearly 2.8 million backward applications of (Res2) in the random scenario while in the sorted scenario there are only half a million. Because of use-check this results only in about 5000 more calls of the CPC prover.

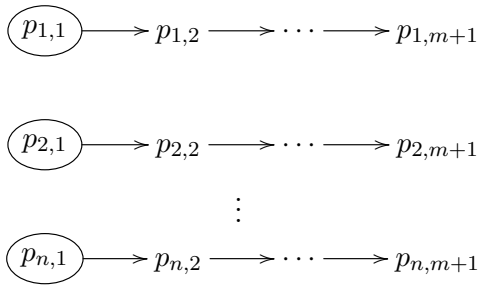
Observation
For this problem we observe that although all residue are necessary, we can avoid a lot of redundancies with to use-check. The impact of it is not as strong as with problems where not all residues are needed. Nevertheless PR2u and PR2gu can still compete with PRcl in the random scenario.
Furthermore this problem confirms an observation of the previous problem, i.e. that PRqs works better if there are few quasi-supports.

6.3.2 Problem 2: Short Chains of Residues

We continue with a simple residue theory that is defined as follows.

$$T_2(n,m) := \{p_{i,1} : 1 \leq i \leq n\} \cup \bigcup_{i=1}^{n} \left\{ \frac{p_{i,j}}{p_{i,j+1}} : 1 \leq j \leq m \right\}$$

In this theory we have n chains of $m+1$ residues. Depicted as a graph $T_2(n,m)$ looks as follows.



The closure of $T_2(n,m)$ contains as in the previous theory all consequents of its residues. In the sorted scenario we use the prerequisite as primary and the consequent as secondary sort key. The sorting of the propositions is done lexicographically according to their indexes, i.e. for $n = m = 3$ we process the residues in the following order:

$$\frac{p_{1,1}}{p_{1,2}}, \frac{p_{1,2}}{p_{1,3}}, \frac{p_{1,3}}{p_{1,4}}, \frac{p_{2,1}}{p_{2,2}}, \frac{p_{2,2}}{p_{2,3}}, \frac{p_{2,3}}{p_{2,4}}, \frac{p_{3,1}}{p_{3,2}}, \frac{p_{3,2}}{p_{3,3}}, \frac{p_{3,3}}{p_{3,4}}$$

Proving $T_2(n,5) \supset p_{1,6} \vee \cdots \vee p_{n,6}$

The first problem for $T_2(n,m)$ is set in a way that we need one of the n chains of residues to prove it.
The problem resembles proving $T_1(n) \supset p_2 \vee \cdots \vee p_n$ but the performances results for the fast provers are turned upside down and three previously slow provers turn out to be fast for one scenario.

Unsteady Provers

Fast in Sorted Scenario In the sorted scenario PR1, PR1g and PR1u show the best performance (together with PR2u and PR2gu). PR1gu lags a bit behind because of the overhead produced by using to improvement methods (c.f. Figure 6.22).
PR1, PR1g, PR1u and PR1gu perform so good because of the sorted processing order. The provers at first skip all residues until they end up in the invalid CPC sequent $p_{1,1}, p_{2,1}, \ldots, p_{n,1} \supset p_{1,6} \vee p_{2,6} \vee \cdots \vee p_{n,6}$. Then

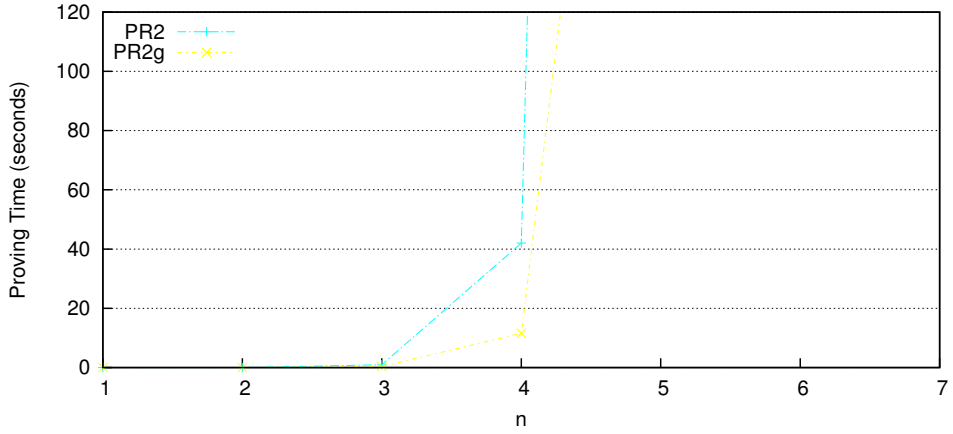


Figure 6.20: Proving time of $T_2(n) \supset p_{1,6} \vee \cdots \vee p_{n,6}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2	168E6	185E6	–	–	201E6
PR2g	16E6	34E6	16E6	–	34E6

Table 6.13: Figures of proving $T_2(5,5) \supset p_{1,6} \vee \cdots \vee p_{5,6}$ (sorted order)

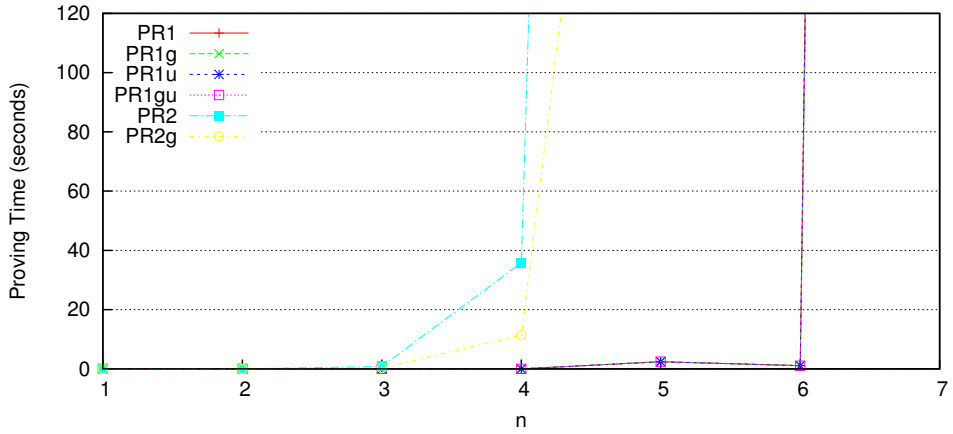


Figure 6.21: Proving time of $T_2(n) \supset p_{1,6} \vee \cdots \vee p_{n,6}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR1/PR1u	357E3	353E3	–	–/0	354E3
PR1g/PR1gu	353E3	353E3	1123	–/0	352E3
PR2	116E6	138E6	–	–	174E6
PR2g	17E6	34E6	17E6	–	34E6

Table 6.14: Figures of proving $T_2(5,5) \supset p_{1,6} \vee \cdots \vee p_{n,6}$ (random order)

they have to backtrack. Because a block of five consecutive residues is sufficient to prove the sequent, they only need to backtrack five levels in the proof search tree. The figures for $n = 2000$ confirm this (c.f. Table 6.15). $T_2(2000, 5)$ contains 10 000 residues and for PR1 and PR1u we have about the same number of applications of (Res1) and 48 applications of (Res2). Most important, we get along with only 80 calls of the CPC prover. The figures for PR1g and PR1gu are similar.

Slow in Random Scenario In the random scenario (c.f. Figure 6.21) the provers based on PR1 drop heavily in performance because the residues that are minimally needed to prove the sequent are no longer in consecutive order. As a consequence the provers mark other residues as proved and therefore have do a lot of backtracking. The figures for $n = 5$ confirm this (c.f. Table 6.14). There are about 350 000 applications of each residue rule and also as many calls of the CPC prover.

Fast Provers
PR2gu, PR2u, PRcl and PRqs perform comparably good. In both scenarios PR2gu and PR2u perform best followed by PRcl and PRqs (c.f. Figure 6.22 and 6.23). We analyze the provers for $n = 2000$ (c.f. Table 6.15 and 6.16).

PRqs shows the same performance in both scenarios. For this problem we have n quasi-supports which can all be extended to a support.

PRcl has for this problem a smaller drop in performance than for the previous problem when changing to random processing order. This is due to the short chains of residues of which the theory is built. Because there are at most 4 residues needed to prove the prerequisite of each residue, the prover has to loop at most 5 times over the residues. Therefore the effect of the random processing order is not as bad as with the previous theory. The figures confirm this observation. There are only twice the number of CPC prover calls in the random scenario compared to the sorted scenario. And this given the fact that for the sorted scenario PRcl shows its best case behavior.

PR2gu and PR2u show the best performance for this problem. This is due to the small number of residues that are minimally needed for the proof.
The sorted scenario leads to the best case behavior of the provers because the five lastly processed residues form a chain that is minimally needed to prove the sequent and in the first call of the CPC prover the algorithm chooses the consequent of the lastly processed residue as principal formula. As a result use-check succeeds in all but the last 5 nodes of the search tree. The figures confirm this (c.f. Table 6.15).
In the random scenario the performance does not drop as dramatically as for the provers based on PR1. This is because with use-check proving the second premise can be omitted in most of the cases. However, the residues that are minimally needed are now to find at arbitrary positions in the search tree.

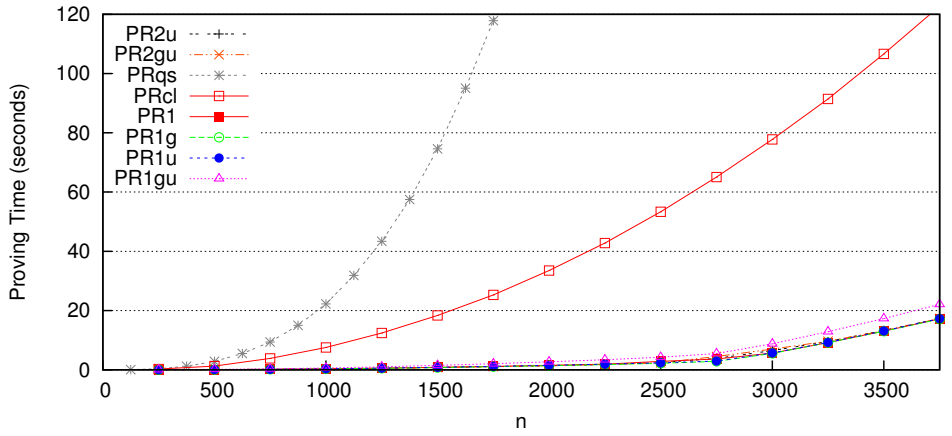


Figure 6.22: Proving time of $T_2(n,5) \supset p_{1,6} \vee \cdots \vee p_{n,6}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR1/PR1u	10E3	48	–	–/0	80
PR1g/PR1gu	10E3	31	10	–/0	37
PR2gu/PR2u	0	10E3	0/–	10E3	6
PRcl	–	–	–	–	10E3
PRqs	–	–	–	–	6

Table 6.15: Figures of proving $T_2(2000,5) \supset p_{1,6} \vee \cdots \vee p_{2000,6}$ (sorted order)

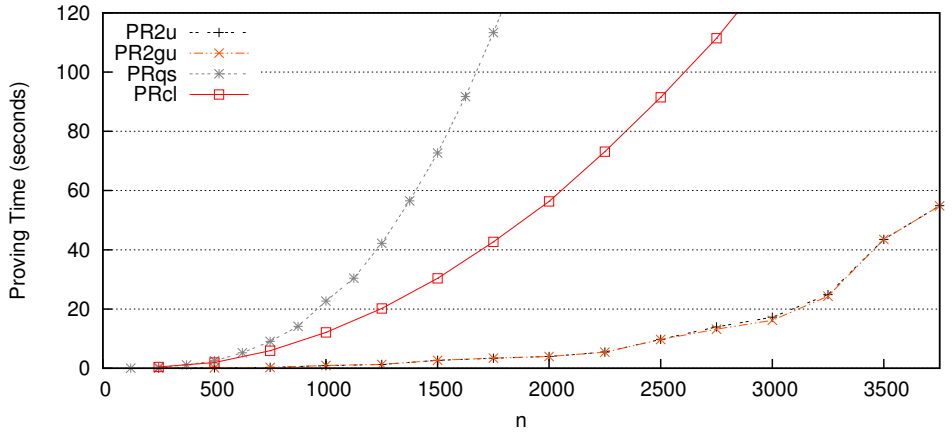


Figure 6.23: Proving time of $T_2(n,5) \supset p_{1,6} \vee \cdots \vee p_{n,6}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	34E3	0/–	34E3	6
PRcl	–	–	–	–	20E3
PRqs	–	–	–	–	6

Table 6.16: Figures of proving $T_2(2000,5) \supset p_{1,6} \vee \cdots \vee p_{n,6}$ (random order)

Therefore backtracking is done closer to the root of the search tree and thus more residue rules have to be processed. Nevertheless, the figures show that because of use-check we still call the CPC prover only 6 times (c.f. Table 6.16).

Observation
For this problem we observe that the provers based on PR1 can compete with the other provers if only few residues are necessary to prove the sequent and if they are processed in a optimal order.
The provers based on PR2u are similar fast here. The reason for this is also because there are only few residues needed for the proof. Compared to the prover based on PR1 the non-optimal processing order of the residues does not have such a heavy impact.
The problem also shows that the drop in performance of PRcl for the random scenario is not so heavy if for every residue's prerequisite there's a support with only little elements, i.e. if we have short chains of residues.

Proving $T_2(n, 5) \supset p_{1,6} \wedge \cdots \wedge p_{n,6}$

The second problem we investigate for the theory $T_2(n, 5)$ is to prove the conjunction $p_{1,6} \wedge \cdots \wedge p_{n,6}$. To prove this sequent all residues of the theory are necessary.

The provers PR2, PR2g and those based on PR1 perform bad. For $n = 5$ all of them exceed the 2 minutes limit by far. We therefore only discuss the results of the other prover (c.f. Figure 6.24 and 6.25). The analysis is done for $n = 500$ (c.f. Table 6.17 and 6.18).

Fast Provers
PRcl shows the same performance as for the previous problem because the calculation of the closure remains the same and proving the succedent when knowing the closure is easy. In both scenarios it performs best among the fast provers.
Although there is now only one minimal quasi-support, the performance of PRqs drops by a factor of about 4. This is because that quasi-support contains all the ending elements of the residue chains. The prover therefore has to calculate the quasi-supports for every residue's precondition. The figure for $T_2(500, 5)$ confirm this. There are 2501 calls of the CPC prover. One to calculate the minimal quasi-supports of $p_{1,6} \wedge \cdots \wedge p_{n,6}$ and the others to calculate the minimal quasi-supports for each residue's prerequisite. If we compare the figures for the sorted scenario with those of PRqs, then we see that we have the same number of calls of the CPC prover. However, PRqs uses the more efficient standard CPC prover while PRqs uses the modified CPC prover that calculates the quasi-supports. Therefore PRqs shows a

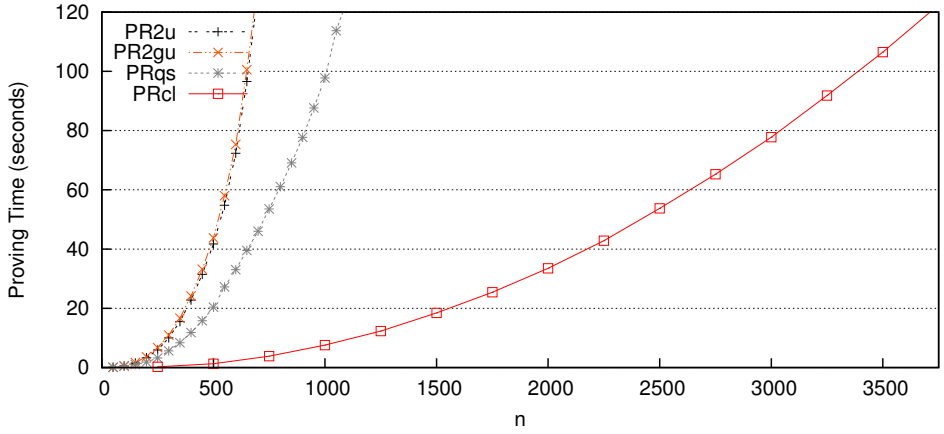


Figure 6.24: Proving time of $T_2(n,5) \supset p_{1,6} \wedge \cdots \wedge p_{n,6}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	3.1E6	0/–	3.1E6	2501
PRcl	–	–	–	–	2501
PRqs	–	–	–	–	2501

Table 6.17: Figures of proving $T_2(500,5) \supset p_{1,6} \wedge \cdots \wedge p_{500,6}$ (sorted order)

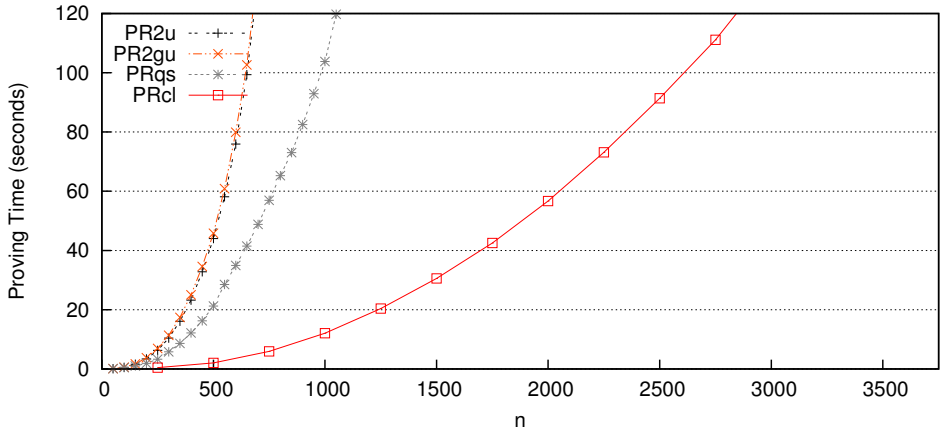


Figure 6.25: Proving time of $T_2(n,5) \supset p_{1,6} \wedge \cdots \wedge p_{n,6}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu/PR2u	0	3.1E6	0/–	3.1E6	2501
PRcl	–	–	–	–	5043
PRqs	–	–	–	–	2501

Table 6.18: Figures of proving $T_2(500,5) \supset p_{1,6} \wedge \cdots \wedge p_{500,6}$ (random order)

much better performance.

PR2gu and PR2u perform worst among the fast provers. The reason becomes obvious if we consult the figures for these provers. A fifth of the residues turn out to be needed in the first call of the CPC prover. Hence a lot of backtracking is required. For $T_2(500, 5)$ we end up in over 3 million applications of (Res2). Although use-check is successful in most of the cases — we end up in only 2501 calls of the CPC prover — the numerous residue rule applications have their impact on the performance of the prover.
In the random scenario it is interesting to see that we have exactly the same figures as in the sorted scenario. The order in which the residues are processed doesn't seems to be of great importance for this problem.

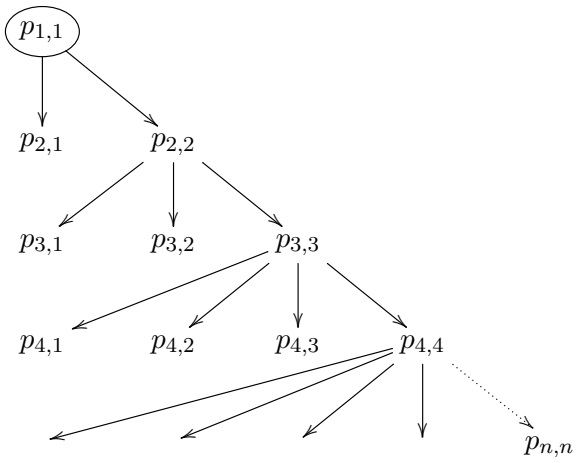
Observation
This problem shows that the performance of PR2gu and PR2u is worse if more residues are needed for the proof of the sequent. It thus confirms the observation made in the previous problem that those provers are fast if there are few residues needed for the proof.

6.3.3 Problem 3: Chain of Residues With Dead Ends

The next theory we inspect is defined as follows.

$$T_3(n) := \{p_{1,1}\} \cup \bigcup_{i=1}^{n-1} \left\{ \frac{p_{i,i}}{p_{i+1,j}} : 1 \leq j < i+1 \right\}$$

This theory defines a chain of residues from $p_{1,1}$ to $p_{n,n}$. Furthermore from each proposition $p_{i,i}$ $(1 \leq i < n)$ there are residues that lead to the propositions $p_{i+1,1}, \ldots p_{i+1,i}$. Since there are no residues that allow us to go further from that proposition we speak of them as dead ends. Depicted as a graph the theory looks as follows.



The closure of $T_3(n)$ consists of $p_{1,1}$ and all its residue's consequents. In the sorted scenario we use the prerequisite as primary and the consequent as secondary sort key. The sorting of the propositions is done lexicographically according to their indexes.

Proving $T_3(n) \supset p_{n,n}$

The problem we investigate for $T_3(n)$ is to prove $p_{n,n}$, i.e. the last element in the chain of residues going from $p_{1,1}$ to $p_{n,n}$.

The provers PR2, PR2g and those based on PR1 perform bad. In both scenarios all of them already exceed the 2 minutes limit for $n = 7$. We therefore omit discussing them.

Fast Provers
In the sorted scenario PRqs performs best followed by PRcl and PR2u together with PR2gu (cf. Figure 6.26 and Table 6.19). We analyze the provers for $n = 100$.

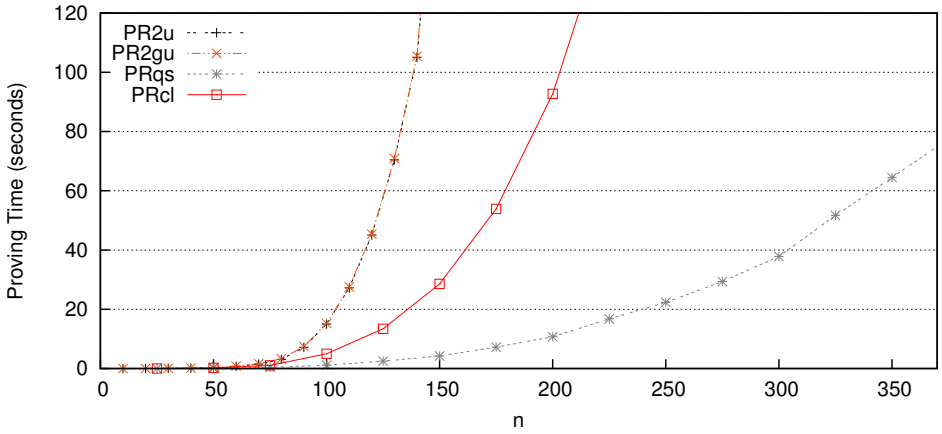


Figure 6.26: Proving time of $T_3(n) \supset p_{n,n}$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu	0	333E3	0	333E3	100
PR2u	0	333E3	–	333E3	100
PRcl	–	–	–	–	5050
PRqs	–	–	–	–	100

Table 6.19: Figures of proving $T_3(100) \supset p_{100,100}$ (sorted order)

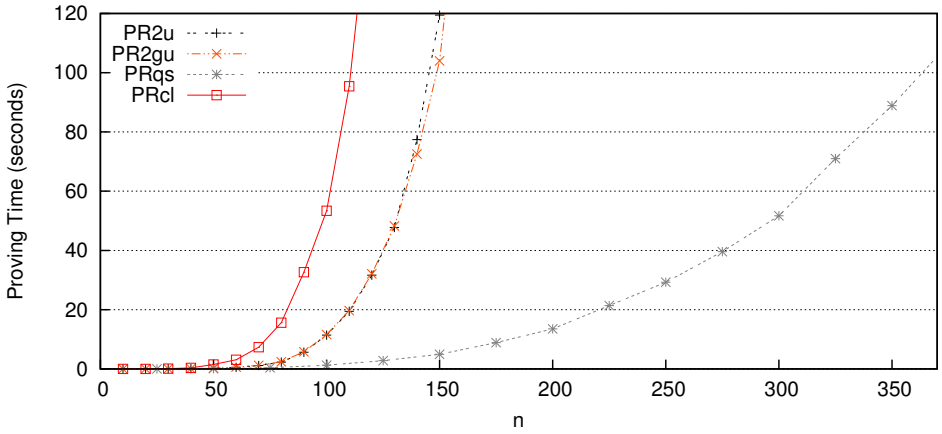


Figure 6.27: Proving time of $T_3(n) \supset p_{n,n}$ (random order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu	0	267E3	0	267E3	100
PR2u	0	267E3	–	267E3	100
PRcl	–	–	–	–	167E3
PRqs	–	–	–	–	100

Table 6.20: Figures of proving $T_3(100) \supset p_{100,100}$ (random order)

The good performance of PRqs has two reasons. Firstly there is only one minimal quasi-support $\{p_{n,n}\}$ for the problem. Secondly there is just one path leading from $p_{1,1}$ to $p_{n,n}$. Therefore all the prover has to do is follow that path backwards. The figures for $n = 100$ confirm this. The prover calls the CPC prover exactly 100 times. Once to calculate the minimal quasi-supports of the sequent that is to prover and 99 times to calculate the quasi-supports of a residue's prerequisite.
PRcl calculates the whole closure and therefore has to verify for each residue whether its prerequisite is in the closure. The sorted order is optimal for this prover, i.e. it has to loop only once over the residues. Nevertheless it can not compete with PRqs. The reason is because the dead ends grow quadratically while the path from $p_{1,1}$ to $p_{n,n}$ grows only linearly with n. The figures confirm the claim about the optimal order. $T_3(100)$ contains 50 049 residues and PRcl calls the CPC prover 50 050 times. The extra call is done to show that the succedent is provable from the closure of $T_3(n)$.
PR2gu and PR2u perform worst among the fast provers. The reason is also to find in the quadratically growing number of residues. Although we can omit proving the second premise in most of the cases, we have to process all of them. This leads to a lot of overhead and thus to a worse performance. The figures confirm this. For $n = 100$ we apply (Res2) over 300 000 times. In all but 99 cases we can omit proving the second premise and end up in calling the CPC prover 100 times, which is optimal for that problem.

In the random scenario PRqs, PR2gu and PR2u show similar good performances. The random processing order even seems to improve their performance a little bit.
PRcl drops in performance. This is because the residues are no longer processed in an optimal order. The prover therefore has to process most of the residues several times. The figures confirm this. For $n = 100$ there are now over 160 000 calls of the CPC prover while in the optimal case there were just about 5000 such calls.

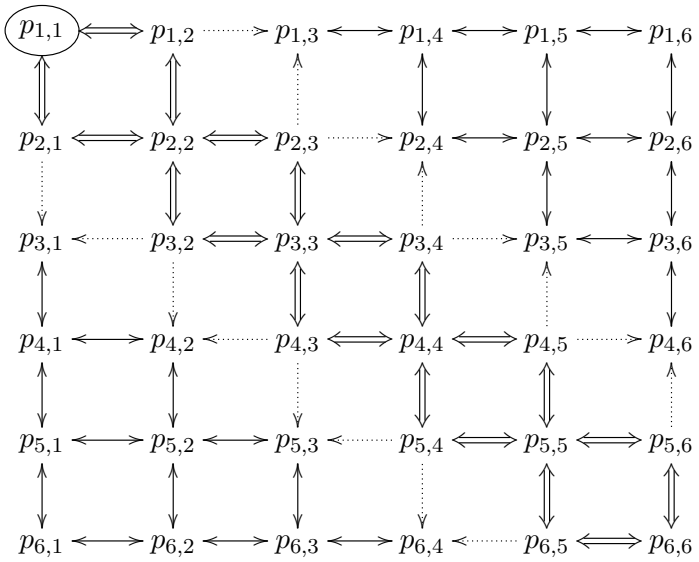
Observation
This problem shows that PRqs outmatches the other provers if there is a single minimal support containing only a small fraction of the residues and most or all the residues prerequisite are in the closure of the theory.
We also see that PRcl depends the most on the order in which the residues are processed.

6.3.4 Problem 4: Grid With Dead Zones

The next theory we inspect is defined as follows.

$$\begin{aligned} T_5(n) := \{p_{1,1}\} &\cup \left\{ \frac{p_{i,j}}{p_{i-1,j}} : 1 < i \leq n, 1 \leq j \leq n \text{ and } j \neq i-2 \right\} \\ &\cup \left\{ \frac{p_{i,j}}{p_{i,j-1}} : 1 \leq i \leq n, 1 < j \leq n \text{ and } j \neq i+2 \right\} \\ &\cup \left\{ \frac{p_{i,j}}{p_{i,j+1}} : 1 \leq i \leq n, 1 \leq j < n \text{ and } j \neq i-2 \right\} \\ &\cup \left\{ \frac{p_{i,j}}{p_{i+1,1}} : 1 \leq i < n, 1 \leq j \leq n \text{ and } j \neq i+2 \right\} \end{aligned}$$

This theory defines a grid consisting mainly of bidirectional and some unidirectional connections.
Following a unidirectional connection on a path starting at $p_{1,1}$ leads into a zone where no path to $p_{n,n}$ exists, we therefore call them dead zones.
There are 2^{n-1} paths of length $2n-2$ from $p_{1,1}$ to $p_{2,2}$. They all run along a diagonal that consists of bidirectional connections. From that diagonal there are unidirectional connections into the two dead zones.
The number of connections in the diagonal grows linearly with n while the number of connections in the dead zone grows quadratically with n.
Depicted as a graph the theory $T_4(6)$ looks as follows. We use thick arrows for the connections in the diagonal, dotted arrows for the connections leading to the dead zones and normal arrows for bidirectional connections inside the dead zones.



The closure of $T_4(n)$ consists of $p_{1,1}$ and all its residue's consequents. In

contrast to the previous theory there is no immediate dead-end as soon as the dead zone is reached.
In the sorted scenario we use the prerequisite as primary and the consequent as secondary sort key. The sorting of the propositions is done lexicographically according to their indexes.

Proving $T_4(n) \supset p_{n,n}$

The problem we investigate for $T_4(n)$ is to prove $p_{n,n}$, i.e. we check whether there's a path from the top left to the down right position in our grid.

Fast Provers
The provers PR2, PR2g and those based on PR1 all perform bad. In both scenarios all of them already exceed the 2 minutes limit for $n = 4$.

Compared to the previous problem, there are now exponentially many minimal supports. PRqs performs best followed by PRcl and PR2u together with PR2gu (cf. Figure 6.28 and Table 6.21).

PRqs performs best because with its strategy it does not enter the dead zones. It tracks back just along the diagonal but has to process most of the residues that span the diagonal due to the bidirectional connections. The figures for $n = 50$ confirm this. There are $8(n-1) = 392$ residues that span the diagonal and the prover requires 390 calls of the CPC prover. But that is little compared to the $4(n^2 - 2n - 2) = 9608$ residue the theory has.
According to the figures the chosen processing order of the residues is optimal for PRcl, i.e. it loops just once over the residues. Nevertheless it performs worse than PRqs because in contrast to PRqs it processes all residues.
PR2gu and PR2u also enter the dead zone in their search and end up with many superfluous rule applications. For $n = 50$ they apply rule (Res2) almost 500 000 times. Because of use-check only 99 calls of the CPC prover are necessary, which is in fact the minimally needed number of calls. Still, the overhead produced by processing that many residues leads to a performance that is not as good as the other two provers.

For the random scenario PRqs performs equally good while PRcl drops in performance and PR2gu and PR2u show an unsteady and poor performance. For PRcl the processing order is no longer optimal, the performance thus drops. The figures are accordingly: In contrast to the optimal case the prover now has to process most of the residues several times and ends up in calling the CPC prover over 140 000 time, that's about 15 times more than in the optimal case (cf. Table 6.22).
Using a random processing order is crucial for PR2gu and PR2u. It takes PR2u over 20 hours to prove the problem for $n = 5$. Because of general improvement PR2gu behaves better. Nevertheless for $n = 25$ it processes

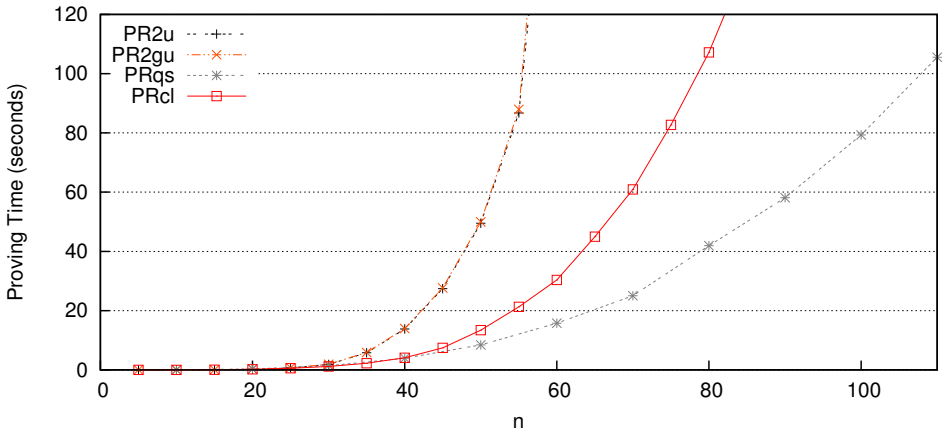


Figure 6.28: Proving time of $T_5(n) \supset$ (sorted order)

	Res1	Res2	gen.imp.	use-check	CPC
PR2gu	0	486E3	0	486E3	99
PR2u	0	486E3	–	486E3	99
PRcl	–	–	–	–	9609
PRqs	–	–	–	–	390

Table 6.21: Figures of proving $T_4(50) \supset p_{50,50}$ (sorted order)

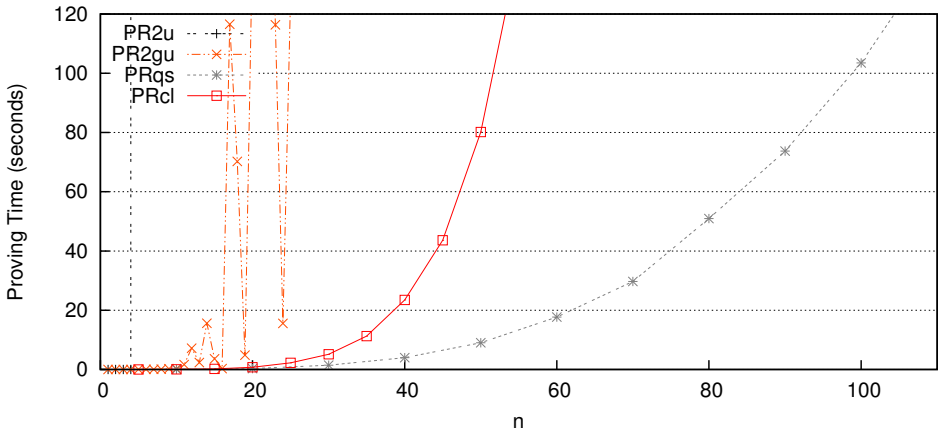


Figure 6.29: Proving time of $T_4(n) \supset p_{n,n}$ (random order)

	n	Res1	Res2	gen.imp.	use-check	CPC
PR2gu	25	65E3	8.8E6	65E3	8.8E6	10E3
PRcl	50	–	–	–	–	142E3
PRqs	50	–	–	–	–	390

Table 6.22: Figures of proving $T_4(n) \supset p_{n,n}$ (random order)

nearly 9 million residue rules and ends up in about 10 000 calls of the CPC prover.

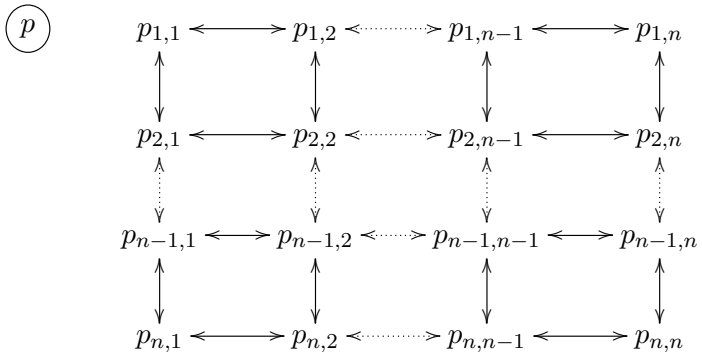
Observation
Besides confirming the observations of the previous problem this problem shows that general improvement can be of use in combination with use-check. In the previous problems the provers PR2u and PR2gu performed always similarly good while for this problem their performance differs for the random scenario.

6.3.5 Problem 5: Grid Without a Start

The next theory we inspect is defined as follows.

$$T_5(n) := \{p\} \cup \bigcup_{i=2}^{n}\bigcup_{j=1}^{n}\left\{\frac{p_{i,j}}{p_{i-1,j}}\right\} \cup \bigcup_{i=1}^{n}\bigcup_{j=2}^{n}\left\{\frac{p_{i,j}}{p_{i,j-1}}\right\} \cup \bigcup_{i=1}^{n}\bigcup_{j=1}^{n-1}\left\{\frac{p_{i,j}}{p_{i,j+1}}\right\} \cup \bigcup_{i=1}^{n-1}\bigcup_{j=1}^{n}\left\{\frac{p_{i,j}}{p_{i+1,1}}\right\}$$

This theory defines a bidirectional grid with a starting point outside the grid. It consists of $4(n^2 - n)$ residues. Depicted as a graph the theory $T_5(n)$ looks as follows.



Because from p we can not deduce the prerequisite of any residue in $T_5(n)$, the closure of $T_5(n)$ is simply $\mathrm{Th}(p)$.
In the sorted scenario we use the prerequisite as primary and the consequent as secondary sort key. The sorting of the propositions is done lexicographically according to their indexes.

Refuting $T_5(n) \supset p_{n,n}$

The problem we investigate for $T_5(n)$ is to refute $p_{n,n}$.

Slow Provers
The provers PR2, PR2g, PR2gu and those based on PR1 all perform bad. In both scenarios they exceed the 2 minutes limit already for $n = 3$ or $n = 4$. Because of the combination of global improvement and use-check PR2gu performs a little bit better and exceeds the 2 minutes limit for $n = 7$.

Fast Provers The picture for PRcl and PRqs is similar for both scenarios. Although both provers call the CPC prover equally often PRcl performs by far better than PRqs (c.f. Figure 6.30 and 6.31).

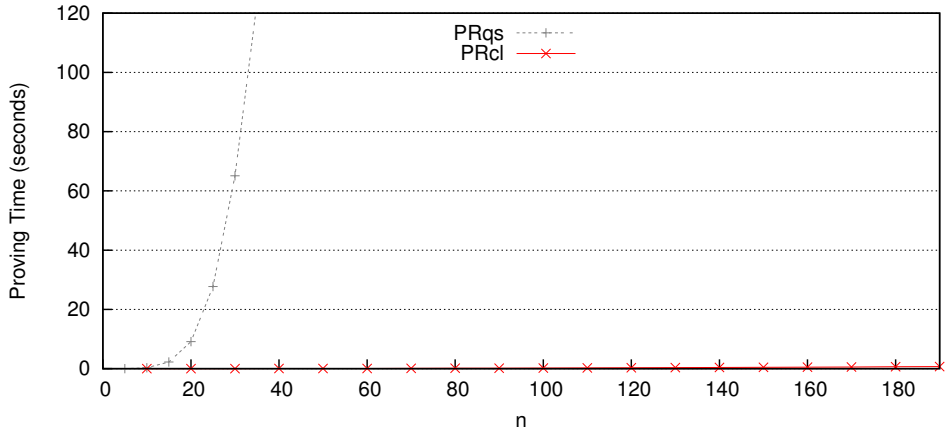


Figure 6.30: Proving time of $T_6(n) \supset$ (sorted order)

	CPC	CPC $\top$	CPC $\bot$
PRcl	1521	0	1521
PRqs	1521	1521	0

Table 6.23: Figures of proving $T_5(20) \supset p_{20,20}$ (sorted order)

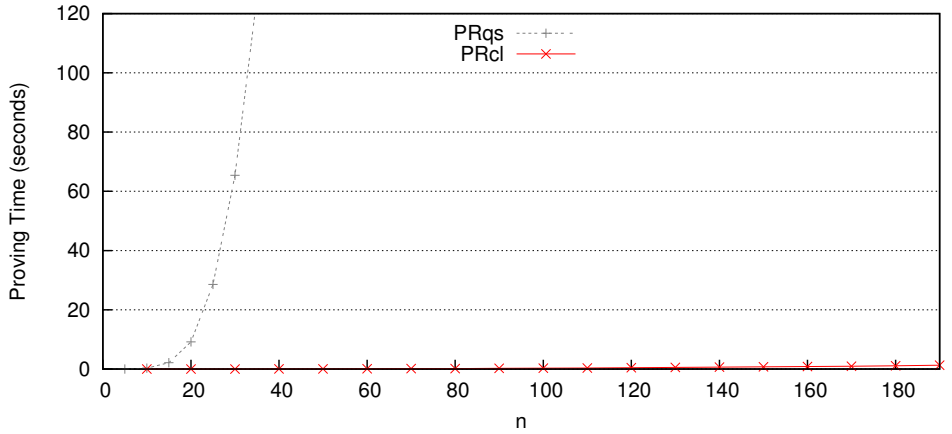


Figure 6.31: Proving time of $T_6(n) \supset$ (random order)

	CPC	CPC $\top$	CPC $\bot$
PRcl	1521	0	1521
PRqs	1521	1521	0

Table 6.24: Figures of proving $T_5(20) \supset p_{20,20}$ (random order)

The difference is easy to explain.
PRcl loops once over the residues to find out that no prerequisite is derivable from p. The sequents it encounters are all of the form $p \supset p_{i,j}$ and easy to refute.
PRqs starts with $p_{n,n}$ and tries to find its minimal supports. To do so it scans through the grid calculating for every residue's prerequisite the minimal quasi-supports only to find out that there is no support for $p_{n,n}$. The sequents it encounters contain n^2 propositions and calculating their minimal quasi-support is much harder than refuting a very simple sequent.

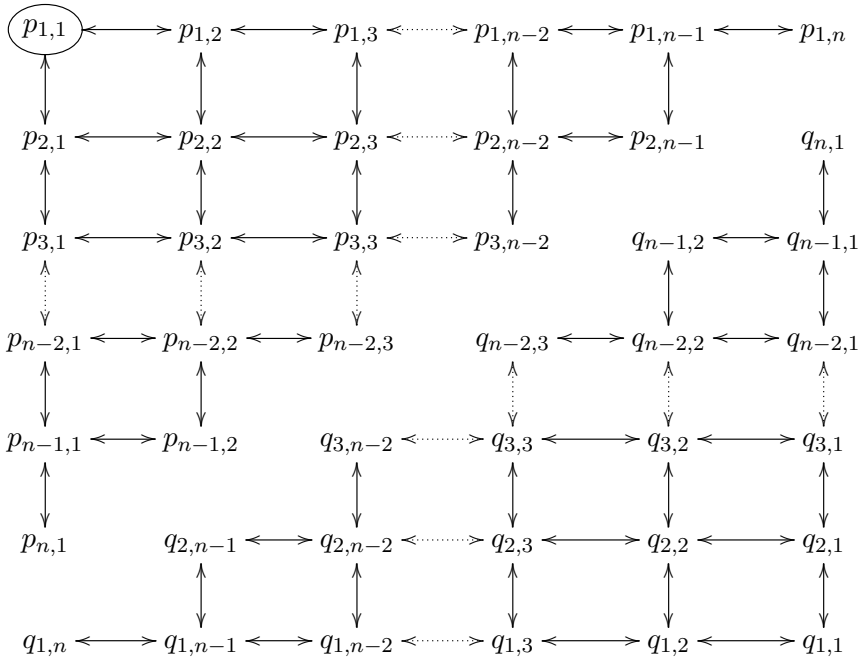
Observation
In the previous two problems we have seen that the strategy of PRqs can be advantageous. This problem shows that this strategy can as well be disadvantageous. However, PRqs performs better than all provers that do backward proof search.

6.3.6 Problem 6: Grid Ripped Apart

The last theory we inspect is defined as follows.

$$
\begin{aligned}
T_6(n) := \{p_{1,1}\} &\cup \bigcup_{i=2}^{n}\bigcup_{j=1}^{n+1-i}\left\{\frac{p_{i,j}}{p_{i-1,j}}\right\} \cup \bigcup_{i=2}^{n}\bigcup_{j=1}^{n+1-i}\left\{\frac{q_{i,j}}{q_{i-1,j}}\right\} \\
&\cup \bigcup_{i=1}^{n}\bigcup_{j=2}^{n+1-i}\left\{\frac{p_{i,j}}{p_{i,j-1}}\right\} \cup \bigcup_{i=1}^{n}\bigcup_{j=2}^{n+1-i}\left\{\frac{q_{i,j}}{q_{i,j-1}}\right\} \\
&\cup \bigcup_{i=1}^{n}\bigcup_{j=1}^{n-i}\left\{\frac{p_{i,j}}{p_{i,j+1}}\right\} \cup \bigcup_{i=1}^{n}\bigcup_{j=1}^{n-i}\left\{\frac{q_{i,j}}{q_{i,j+1}}\right\} \\
&\cup \bigcup_{i=1}^{n}\bigcup_{j=1}^{n-i}\left\{\frac{p_{i,j}}{p_{i+1,1}}\right\} \cup \bigcup_{i=1}^{n}\bigcup_{j=1}^{n-i}\left\{\frac{q_{i,j}}{q_{i+1,1}}\right\}
\end{aligned}
$$

This theory defines a bidirectional grid that is ripped apart. It consists of $4(n^2 - n)$ residues. Depicted as a graph the theory $T_6(n)$ looks as follows.



The closure of $T_6(n)$ is $\mathrm{Th}(\{p_{i,j} : 1 \leq i, j \leq n \text{ and } i + j \leq n + 1\})$.
In the sorted scenario we use the prerequisite as primary and the consequent as secondary sort key. The sorting of the propositions is done primary according to their indexes and secondary according to their base name.

Refuting $T_6(n) \supset q_{1,1}$

The problem we investigate for $T_6(n)$ is to refute $q_{1,1}$. It is constructed in a way that a bottom up and top down strategy lead to a similar amount of work.

Slow Provers
The provers PR2, PR2g, PR2u and those based on PR1 all perform bad. In both scenarios they exceed the 2 minutes limit already for $n = 3$, $n = 4$ or $n = 5$. Because of the combination of global improvement and use-check PR2gu performs a little bit better and exceeds the 2 minutes limit for $n = 7$.

Fast Provers
In the sorted scenario PRcl is clearly faster than PRqs (c.f. Figure 6.32). This is mainly because the sorting order is close to optimal for PRcl. It loops only twice through the residues and needs to process only half of the residues in the second pass. The figures for $n = 50$ confirm this (c.f. Table 6.25). There are 14 701 calls of the CPC prover of which 4900 succeed.
PRqs has less calls of the CPC prover but it uses a CPC prover that calculates the minimal quasi-supports, it is slower than PRqs.

In the random scenario the provers are closer together because the processing order is now no longer optimal for PRcl (c.f. Figure 6.33). Nevertheless PRcl is still faster than PRqs but the gap between the two provers is smaller than for the sorted scenario. The figures confirm this observation (c.f. Table 6.26). While PRqs is stable and has the same number of calls to the CPC prover, PRcl now calls the CPC prover about 150 000 times.

Observation
When presenting a problem that leads to a similar amount of work for PRqs and PRcl both provers show a good performance. The performance of PRcl depends on the order in which the residues are processed. Nevertheless it performs in both scenarios better than PRqs.

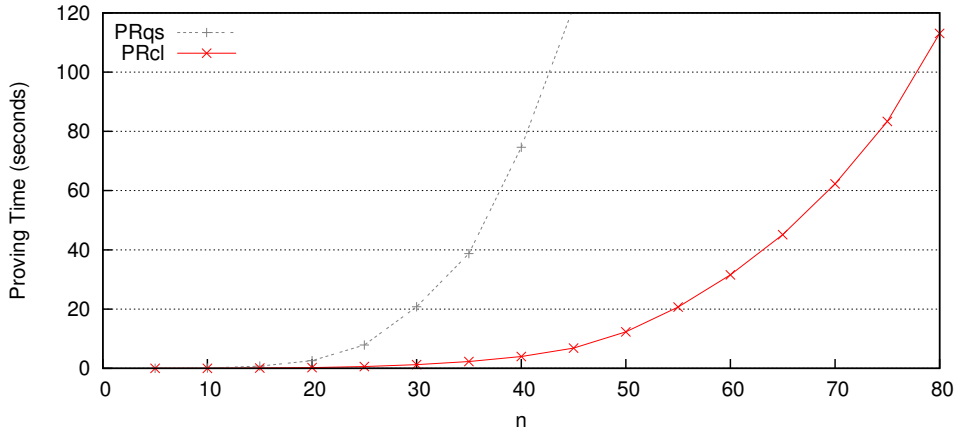


Figure 6.32: Proving time of $T_6(50) \supset q_{1,1}$ (sorted order)

	CPC	CPC $\top$	CPC $\bot$
PRcl	14 701	4900	9801
PRqs	4901	4901	0

Table 6.25: Figures of proving $T_6(20) \supset q_{1,1}$ (sorted order)

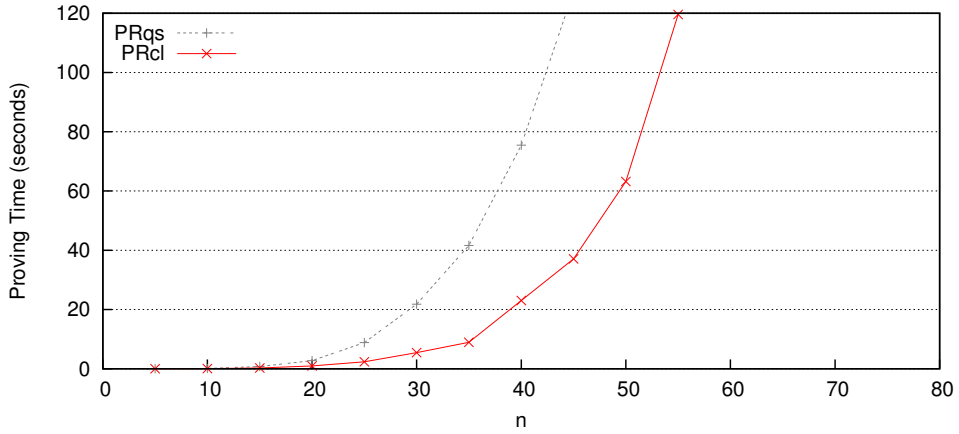


Figure 6.33: Proving time of $T_6(n) \supset q_{1,1}$ (random order)

	CPC	CPC $\top$	CPC $\bot$
PRcl	150E3	4900	145E3
PRqs	4901	4901	0

Table 6.26: Figures of proving $T_6(50) \supset q_{1,1}$ (random order)

6.3.7 Conclusion

The experimental results show that the provers based on PR1 are generally slow. They only perform good if little residues are needed for the proof and the residues are processed in an optimal order.

The provers PR2 and PR2g perform bad for all problems we have examined. General improvement is useful for some problems but its impact is rather weak.

PR2gu and PR2u show good performances for most of the valid problems we have inspected. The good performance is thereby mainly due to use-check. The only valid problem where they perform poor is the random scenario of problem 4. They perform best if only few of the residues are needed for the proof and show their best behavior if these residues are processed lastly. Both provers show a rather stable behavior when changing the processing order of the residues. For the two harder to solve invalid problems presented at the end they show a poor performance.

PRqs performs good for most problems. For all but its worst case scenario (problem 5) it can compete with the other provers and performs best for several of the examined problems. It performs best if few of the residues are to be considered when calculating the minimal support. Furthermore, because the underlying CPC prover uses sorted containers the processing order of the residue has little influence on its performance.

PRcl performs good for all problems and can compete with the other provers. It performs best for several of the examined problems but is often heavily influenced by the order in which the residues are processed.

In summary we observe that PR2gu, PR2u, PRqs and PRcl are good enough to check the validity of a residue sequent. The latter two seem to perform better for the less trivial problems. And from these two, PRqs proves to be the most stable regarding the processing order of the residues.

Chapter 7

Proof Search in Default Logic

For automatic proving in propositional default logic we investigate two approaches. We start with backward proof search in credulous and skeptical default logic and refine the presented algorithms step by step. Then we present an algorithm to compute extensions to have a comparison with extension based proving. We end the chapter with some experimental results.

7.1 Proof Search in Credulous Default Logic

In this section we investigate backward proof search for credulous default logic. We start with a simple prover that just backward applies the rules of cPDC and then improve it by avoiding obvious redundancies. Afterward we further improve our algorithm by a preprocessing step to narrow the search tree. For this improvement we use the minimal supports of the corresponding residue theory. We close the section with the introduction of an improvement technique that is similar to the general improvement of the residue prover.

7.1.1 A Simple Prover

A simple backward proof search algorithm for credulous default logic can easily be given. The rules (cD1), (cD2) and (cD3) are invertible while (cD4) and (cD5) are not. Backtracking is thus only necessary for the latter two rules.

Lemma 7.1 ((cD1), (cD2) and (cD3) are invertible)
The rules (cD1), (cD2) and (cD3) are invertible.

Proof. The claims follow directly from the proof of soundness and completeness of cPDC. □

Remark 7.2 ((cD4) and (cD5) are not invertible)
The rules (cD4) and (cD5) are not invertible.

For (cD4) consider the default sequent $\emptyset; A, \frac{A : B}{B} \supset B$. It is credulously valid while the corresponding premise $\mathbf{L}\neg B; A \supset B$ is not.

For (cD5) consider the default sequent $\emptyset; A, \frac{\top : \neg A}{\neg A} \supset A$. It is credulously valid while the corresponding premise $\neg\mathbf{L}A; A, \frac{\top}{\neg A} \supset A$ is not.

It is obvious that the order in which we process the defaults is not important. We therefore can give a simple algorithm for backward proof search in credulous default logic (see Algorithm 16).

The algorithm consists of three functions.

credDefProvable initiates the proof search. The given arguments are a default theory $\langle W, D\rangle$ and the formula A that is to prove. The function first splits D into residues R and proper defaults D' (lines 2+3) and then starts proof search by calling cD45 (line 4).

cD45 applies the rules (cD4) and (cD5). The given argument is a default sequent $\Sigma; W, R, D \supset A$, where R holds the residues and D the proper defaults of the sequent.
The function first checks whether any proper defaults are left in the given sequent (line 6). If not it calls cD123 to apply the rules (cD1), (cD2) and (cD3). Otherwise it selects a proper default δ, applies (cD4) with δ as principal formula and recursively calls cD45 with the corresponding premise (lines 9–13). If necessary backtracking is done by using a different justifications of δ as active formula in the premise. If none of the (cD4) rule applications succeeds rule (cD5) is applied (lines 15–17).

cD123 applies the rules (cD1), (cD2) and (cD3). Its arguments is a default sequent that contains only residues. For simplicity the function does not recursively call itself but just loops over the constraints Σ and tries to verify them (lines 21–25). If all constraints are fulfilled the validity of the succedent A is verified (line 26).

Avoiding Obvious Redundancies

The loop on lines 11–14 in function cD45 produces redundant problems. Consider the default sequent $\Sigma; W, D, \delta \supset A$ with $\text{jus}(\delta) := \{B_1, \ldots, B_n\}$.

Algorithm 16 Simple proof search for credulous default logic

1: **function** CREDDEFPROVABLE($\langle W, D\rangle$, A)
2: $R := \{\delta \in D : \mathrm{jus}(\delta) = \emptyset\}$
3: $D' := \{\delta \in D : \mathrm{jus}(\delta) \neq \emptyset\}$
4: **return** cD45$(; W, R, D' \supset A)$

5: **function** cD45$(\Sigma; W, R, D \supset A)$
6: **if** $D = \emptyset$ **then** ▷ only residues left, apply (cD2) and (cD3)
7: result := cD123$(\Sigma; W, R \supset A)$
8: **else** ▷ have proper defaults, apply (cD4), (cD5)
9: choose $\delta \in D$ and let $D' := D \setminus \{\delta\}$
10: result := false
11: **for** $B \in \mathrm{jus}(\delta)$ **do** ▷ apply (cD4) for each justification
12: $\Sigma' := \Sigma \cup \{\mathbf{L}\neg B\}$
13: result := cD45$(\Sigma'; W, D', R \supset A)$
14: **if** result **then break** ▷ exit loop on success
15: **if not** result **then** ▷ apply (cD5)
16: $\Sigma' := \Sigma \cup \{\neg\mathbf{L}\neg B : B \in \mathrm{jus}(\delta)\}$
17: result := cD45$\left(\Sigma'; W, D', R \cup \left\{\frac{\mathrm{pre}(\delta)}{\mathrm{con}(\delta)}\right\} \supset A\right)$
18: **return** result

19: **function** cD123$(\Sigma; W, R \supset A)$
20: result = true
21: **for** $\sigma \in \Sigma$ **do**
22: **if** σ is of the form $\mathbf{L}\neg B$ **then** ▷ left premise of (cD2)
23: result := result **and** RESPROVABLE$(W, R \supset \neg B)$
24: **else** ▷ left premise of (cD3)
25: result := result **and** RESREFUTABLE$(W, R \supset \neg B)$
26: **return** result **and** RESPROVABLE$(W, R \supset A)$ ▷ right premise

In the worst case the loop leads to the following function calls.

$\textsc{cD45}(\Sigma, \mathbf{L}\neg B_1; W, D \supset A)$
$\textsc{cD45}(\Sigma, \mathbf{L}\neg B_2; W, D \supset A)$
$\vdots$
$\textsc{cD45}(\Sigma, \mathbf{L}\neg B_n; W, D \supset A).$

Besides checking for each extension of $\langle W, D\rangle$ whether it fulfills $\mathbf{L}\neg B_i$ for $1 \leq i \leq n$ we also check n times instead of just once whether it fulfills Σ.

This redundancy can be avoided if we shift that loop down the search tree to the nodes where only residues are left. An easy way to do this is to classify the processed defaults according to the applied rule into the sets D_4 and D_5. The set of constraints is thereby no longer necessary since we can construct the constraints from D_4 and D_5. For $\delta_4 \in D_4$ we know that $\mathbf{L}\neg B$ must be fulfilled for an arbitrary $B \in \text{con}(\delta_4)$. For $\delta_5 \in D_5$ we know that $\neg\mathbf{L}\neg B$ must be fulfilled for all $B \in \text{con}(\delta_5)$, that is $\neg\mathbf{L}\neg B$ must be fulfilled for all $B \in \text{con}(D_5)$. The corresponding algorithm is given in Algorithm 17.

credDefProvable' initiates the proof search. The function splits the given defaults into residues and proper defaults and then starts proof search by calling cD45'.

The residues of D are not passed separately to cD45'. Since they have no justifications it is correct to regard them as defaults for which rule (cD5) has been applied.

cD45' applies the rules (cD4) and (cD5). Its argument is a default sequent without constraints but with a disjoint partition (D, D_4, D_5) of the defaults. D holds the defaults that are yet to be processed, D_4 and D_5 hold the defaults for which (cD4) and (cD5) have been applied.

The function is similar to cD45 of Algorithm 16 but when applying (cD4) is does not loop over the justification of the chosen default but simply moves the chosen default to D_4 (line 10). Looping is later done in function cD123'. Applying (cD5) is done in a similar way (line 12).

cD123' applies the rules (cD1), (cD2) and (cD3). Its argument is a default sequent without constraints where the defaults are classified into two sets according to the rule that has been applied to them. The function first creates the set of residues R_5 according to the classification (line 15). Then it creates and verifies the constraints. For each default $\delta_4 \in D_4$ it checks whether $\mathbf{L}\neg B$ is fulfilled for one $B \in \text{jus}(\delta_4)$ (lines 17–21) and for each default $\delta_5 \in D_5$ it checks whether $\neg\mathbf{L}\neg B$ is fulfilled for all $B \in \text{jus}(\delta_5)$ (lines 22–23). In the end it verifies the succedent, i.e. whether $W, R_5 \supset A$ is valid.

Algorithm 17 Partition based proof search for credulous default logic

1: **function** CREDDEFPROVABLE'($\langle W, D\rangle$, A)
2: $R := \{\delta \in D : \text{jus}(\delta) = \emptyset\}$
3: $D' := \{\delta \in D : \text{jus}(\delta) \neq \emptyset\}$
4: **return** CD45'$(W, D', \emptyset, R \supset A)$

5: **function** CD45'$(W, D, D_4, D_5 \supset A)$
6: **if** $D = \emptyset$ **then** ▷ only residues left, apply (cD2) / (cD3)
7: result := CD123'$(W, D_4, D_5 \supset A)$
8: **else** ▷ proper defaults left, apply (cD4) / (cD5)
9: choose $\delta \in D$ and let $D' := D \setminus \{\delta\}$
10: result := CD45'$(\ W, D', D_4 \cup \{\delta\}, D_5 \supset A\ )$ ▷ (cD4)
11: **if not** result **then**
12: result := CD45'$(\ W, D', D_4, D_5 \cup \{\delta\} \supset A\ )$ ▷ (cD5)
13: **return** result

14: **function** CD123'$(W, D_4, D_5 \supset A)$
15: $R_5 := \left\{ \dfrac{\text{pre}(\delta)}{\text{con}(\delta)} : \delta \in D_5 \right\}$ ▷ residues from (cD5)
16: result := true
17: **for** $\delta \in D_4$ **do** ▷ check if positive constraints are fulfilled
18: result := false ▷ at least one constraint must hold
19: **for** $B \in \text{jus}(\delta)$ **do**
20: result := result **or** RESPROVABLE$(W, R_5 \supset \neg B)$
21: **if not** result **then break**
22: **for** $B \in \text{jus}(D_5)$ **do** ▷ check if negative constraints are fulfilled
23: result := result **and** RESREFUTABLE$(W, R_5 \supset \neg B)$
24: **return** result **and** RESPROVABLE$(W, R_5 \supset A)$

7.1.2 Preprocessing

To prove the default sequent $; W, D \supset A$ the recursion done in Algorithm 17 more or less checks for each subset D_5 of D whether cD123'$(W, D \backslash D_5, D_5 \supset A)$ is successful. With a preprocessing step we try to limit the subsets D_5 of D that we have to consider. It bases on the application of (cD1). Given $D_5 \subseteq D$, and with it $R_5 := \{\text{pre}(\delta)/\text{con}(\delta) : \delta \in D_5\}$, we verify in each call of cD123' whether $W, R_5 \supset A$ is valid. Thereby R_5 is a subset of $R := \{\text{pre}(\delta)/\text{con}(\delta) : \delta \in D\}$. We know that $W, R_5 \supset A$ is only valid if R_5 contains a minimal support of $\langle W, R\rangle$ for A. The set of defaults that corresponds to a minimal support is called a minimal requirement.

Definition 7.3 (minimal requirement)
Let $\langle W, D\rangle$ be a default theory, $A \in \mathcal{L}$ and $R := \left\{\frac{\text{pre}(\delta)}{\text{con}(\delta)} : \delta \in D\right\}$.
A set D' is called a *minimal requirement* of A for $\langle W, D\rangle$ if $D' \subseteq D$ and $R' := \left\{\frac{\text{pre}(\delta)}{\text{con}(\delta)} : \delta \in D'\right\}$ is a minimal support of A for $\langle W, R\rangle$.

The idea we follow is to calculate the minimal requirements of A for $\langle W, D\rangle$ and use them to omit applying (cD4). Suppose that $D_{\min}$ is such a minimal requirement. Then the corresponding set of residues of $R_{\min}$ is a minimal support of A. We hence know that proving A will be successful if we apply (cD5) on the elements of $D_{\min}$. For the previous algorithm this means that we can classify the elements of a minimal requirement as being processed by (cD5) before we even start the proof search. The corresponding algorithm is given in Algorithm 18.

credDefProvableMR initiates the proof search. Its difference to credDefProvableMR is that it calculates the minimal requirements of A for $\langle W, D\rangle$ and then uses them to preselect them as being processed by (cD5) before initiating the proof search.

cD45MR is similar to cD45'. In contrast to it the formula A that is to prove is not passed as argument. This is because we know that the residues R_5 computed from D_5 in cD123' are sufficient to prove A.

cD123MR is similar to cD123'. For the same reason as given above is does not take the formula A that is to prove as argument. As a consequent trying to prove A is omitted.

Calculating the Minimal Requirements

In this section we discuss how to calculate minimal requirements.
According to its definition, calculating the minimal requirements of A for $\langle W, D\rangle$ is almost equivalent to calculating the minimal supports of A for

Algorithm 18 Proof search for credulous default logic using minimal requirements

```
1:  function credDefProvableMR(; W, D ⊃ A)
2:      D_5 := {δ ∈ D : jus(δ) = ∅}
3:      D' := {δ ∈ D : jus(δ) ≠ ∅}
4:      D_min := minRequirements(⟨W, D⟩, A)
5:      for D_min ∈ D_min do
6:          result := result or cD45MR(W, D' \ D_min, ∅, D_5 ∪ D_min)
7:      return result

8:  function cD45MR(W, D, D_4, D_5)
9:      if D = ∅ then                          ▷ only residues left, apply (cD2) / (cD3)
10:         result := cD123MR(W, D_4, D_5)
11:     else                                   ▷ proper defaults left, apply (cD4) / (cD5)
12:         choose δ ∈ D and let D' := D \ {δ}
13:         result := cD45MR(W, D', D_4 ∪ {δ}, D_5)                    ▷ (cD4)
14:         if not result then
15:             result := cD45MR(W, D', D_4, D_5 ∪ {δ})                ▷ (cD5)
16:     return result

17: function cD123MR(W, D_4, D_5)
18:     R_5 := { pre(δ)/con(δ) : δ ∈ D_5 }                             ▷ residues from (cD5)
19:     result := true
20:     for δ ∈ D_4 do                         ▷ check if positive constraints are fulfilled
21:         result := false                        ▷ at least one constraint must hold
22:         for B ∈ jus(δ) do
23:             result := result or resProvable(W, R_5 ⊃ ¬B)
24:         if not result then break
25:     for B ∈ jus(D_5) do                    ▷ check if negative constraints are fulfilled
26:         result := result and resRefutable(W, R_5 ⊃ ¬B)
27:     return result
```

$\langle W, R\rangle$, where $R = \{\text{pre}(\delta)/\text{con}(\delta) : \delta \in D\}$.
The difference is that two minimal requirements D'_1 and D'_2 may have the same corresponding minimal support R'. This is because two default rules δ_1 and δ_2 may only differ in their justifications and are therefore interchangeable in a minimal requirement.
Since constructing the minimal requirements of A for $\langle W, D\rangle$ from the minimal supports of A for $\langle W, R\rangle$ is easy we only discuss how to compute the minimal supports.

The following proposition is central to our algorithm for computing minimal supports.

Proposition 7.4 (computing minimal supports)
Let R_A be a minimal support of A for $\langle W, R\rangle$.
Then there exists a minimal quasi-support $R'_A = \{\delta_1, \ldots, \delta_n\}$ of A for $\langle W, R\rangle$ and for $1 \leq i \leq n$ minimal supports R_i of $\text{pre}(\delta_i)$ for $\langle W, R \setminus \{\delta_i\}\rangle$ such that $R_A = \bigcup_{i=1}^{n} R_i \cup R'_A$.

Proof. Let R_A be a minimal support of A for $\langle W, R\rangle$ (1). From Lemma 6.32 we know that $\text{pre}(\delta) \in \text{Cl}(W, R_A) \subseteq \text{Cl}(W, R)$ for $\delta \in R_A$ (i).
Let $R'_A = \{\delta_i, \ldots, \delta_n\}$ be a minimal quasi-support of A for $\langle W, R_A\rangle$ (2). Then $R'_A \subseteq R_A$ (ii) and R'_A is also a minimal quasi-support of A for $\langle W, R\rangle$ (iii).
Let $\delta \in R'_A$. With (i) and (ii) we obtain from Lemma 5.14.4 that $\text{pre}(\delta) \in \text{Cl}(W, R_A \setminus \{\delta\})$. Hence $R_A \setminus \{\delta\}$ is a support of $\text{pre}(\delta)$ for $\langle W, R_A \setminus \{\delta\}\rangle$ and thus there exists a minimal support R_δ of $\text{pre}(\delta)$ for $\langle W, R \setminus \{\delta\}\rangle$ with $R_\delta \subseteq R_A \setminus \{\delta\}$.
Now for $1 \leq i \leq n$ let R_i be a minimal support of $\text{pre}(\delta_i)$ for $\langle W, R \setminus \{\delta_i\}\rangle$ with $R_i \subseteq R_A \setminus \{\delta_i\}$ (iv). Let $R_{1n} := \bigcup_{i=1}^{n} R_i \cup R'_A$. From (ii) and (iv) we know $R_{1n} \subseteq R_A$ (v). Furthermore we know from (2) and (iv) that $A \in \text{Cl}(W, R_{1n})$. With (v) and (1) we thus obtain $R_A = R_{1n}$. □

Proposition 7.4 gives us a recipe how to compute the minimal supports of A for $\langle W, R\rangle$ from the corresponding minimal quasi-supports and the minimal supports of the residues in those minimal quasi-supports.
We have seen that minimal quasi-supports correspond to L-minimal sequents and have given an algorithm to compute them (cf. Algorithm 15 on page 180). Furthermore we know that if $\emptyset$ is a minimal quasi-support then it is also a minimal support. This gives us the recursion base to compute minimal supports.
We hence have everything to write down an algorithm to compute the minimal supports of A for $\langle W, R\rangle$ (cf. Algorithm 19).

The algorithm first computes the minimal quasi-supports of A for $\langle W, R\rangle$ (line 2).

If there exists no minimal quasi-supports or the only minimal quasi-support is the empty set, then we're done and can return the result (lines 3–4).
Otherwise we recursively calculate for each residue δ that is part of a minimal quasi-support the minimal supports $\mathrm{ms}[\delta]$ of $\mathrm{pre}(\delta)$ for $\langle W, R \setminus \{\delta\}\rangle$ (lines 5–6). The recursion is known to terminate because the set of residues is reduced with each step.
From the recursively calculated minimal supports $\mathrm{ms}[\delta]$ we then calculate according to Proposition 7.4 the set of supports $\mathrm{S}[A]$ of which we know that it contains all minimal supports of A for $\langle W, R\rangle$ (lines 8–12). From $\mathrm{S}[A]$ we remove the non-minimal elements and obtain like this the minimal supports of A for $\langle W, R\rangle$.

Algorithm 19 Computing minimal supports

```
 1: function minSupports(⟨W, R⟩, A)
 2:     MQS := minQuasiSupports(⟨W, R⟩, A)
 3:     if MQS = ∅ or MQS = {∅} then                    ▷ recursion base
 4:         return MQS
 5:     for δ ∈ ⋃MQS do      ▷ calculate min. supports of residues in MQS
 6:         ms[δ] := minSupports(⟨W, R \ {δ}⟩, pre(δ))
 7:     S[A] := ∅
 8:     for mqs ∈ MQS do          ▷ extend min. quasi supports to supports
 9:         S[mqs] := mqs
10:         for δ ∈ mqs do
11:             S[mqs] := {S[mqs] ∪ ms : ms ∈ ms[δ]}
12:         S[A] := S[A] ∪ S[mqs]
13:     MS := {s ∈ S : s minimal in S regarding ⊆}
14:     return MS
```

7.1.3 Residual Improvement

The idea we follow with residual improvement takes place in the function cD123' and cD123MR, respectively. There we calculate R_5 from D_5. Depending on whether a default δ is in D_4 or D_5 we have the following cases.

$\delta \in D_4$: Then we check if $W, R_5 \supset \neg B$ is provable for some $B \in \mathrm{jus}(\delta)$.

> If this succeeds then we know that is also succeeds for a set of residues R_5' with $R_5 \subseteq R_5'$, i.e. for a partition (D_4', D_5') with $D_5 \subseteq D_5'$.
>
> If this fails then we know that is also fails for a set of residues R_5 with $R_5' \subseteq R_5$, i.e. for a partition (D_4', D_5') with $D_5' \subseteq D_5$.

$\delta \in D_5$: Then we check if $W, R_5 \supset \neg B$ is refutable for all $B \in \mathrm{jus}(\delta)$.

If this succeeds then we know that it also succeeds for a set of residues R'_5 with $R'_5 \subseteq R_5$, i.e. for a partition (D'_4, D'_5) with $D'_5 \subseteq D_5$.

If this fails then we know that it also fails for a set of residues R'_5 with $R_5 \subseteq R'_5$, i.e. for a partition (D'_4, D'_5) with $D_5 \subseteq D'_5$.

The two cases are dual to each other. That is $W, R_5 \supset \neg B$ is provable for some $B \in \text{jus}(\delta)$ iff $W, R_5 \supset \neg B$ is not refutable for all $B \in \text{jus}(\delta)$. We therefore treat the two cases in a dual sense.

The idea of residual improvement is to first check according to the cases above whether the provability or refutability of a residue sequent is known from previously gained results.

The function cD123MRG given in Algorithm 20 illustrates proof search based on minimal requirement and residual improvement.

cD123MRG loops over the defaults of D_4 and D_5 and uses the function verifySkippedDefault to check the constraints produced by the default.

verifySkippedDefault verifies whether the given default δ is skippable for the given residue theory $\langle W, R\rangle$. We say that a default is skippable for $\langle W, R\rangle$, if one of its justifications is not fulfilled in $\langle W, R\rangle$.

The function uses the global maps skippable[] and nonskippable[]. They map from a default δ to a set of sets of residues $\{R_1, \ldots, R_n\}$ and hold the information for which set of residues R_i a default's justifications are know to be not met and met in $\langle W, R_i\rangle$, respectively. Hence if $R \in$ skippable$[\delta]$ then $W, R \supset \neg B$ is provable for some $B \in \text{jus}(\delta)$. If $R \in$ nonskippable$[\delta]$ then $W, R \supset \neg B$ is not provable for all $B \in \text{jus}(\delta)$.

The function first checks whether R is a superset of a set R' of which we know that the given default is skippable for $\langle W, R'\rangle$ (lines 10–11). If it is then we know that the given default is also skippable for the given theory and can return with success.

The function then checks whether R is a subset of a set R' of which we know that the given default is not skippable for $\langle W, R'\rangle$ (lines 12–13). If it is then we know that the given default is also not skippable for the given theory and can return with failure.

If the previous two checks did not reveal any information the function verifies whether all positive constraints derived from δ are met (lines 15–16) and caches the result accordingly (lines 17–20).

verifyPosConstraint verifies whether the positive constraint $\mathbf{L}A$, given by the formula A, is fulfilled in the given residue theory $\langle W, R\rangle$.

Algorithm 20 Residual improvement for proof search in credulous default logic

```
1: function cD123MRG(W, D_4, D_5)
2:     R_5 := { pre(δ)/con(δ) : δ ∈ D_5 }
3:     result := true
4:     for δ ∈ D_4 do
5:         result := result and verifySkippedDefault(W, R_5, δ)
6:     for δ ∈ D_5 do
7:         result := result and not verifySkippedDefault(W, R_5, δ)
8:     return result

9: function verifySkippedDefault(W, R, δ)
10:    if {R' ∈ skippable[δ] : R' ⊆ R} ≠ ∅ then
11:        return true
12:    if {R' ∈ nonskippable[δ] : R' ⊇ R} ≠ ∅ then
13:        return false
14:    result := false
15:    for B ∈ jus(δ) do
16:        result := result or verifyPosConstraint(W, R, ¬B)
17:    if result then
18:        skippable[δ] += R
19:    else
20:        nonskippable[δ] += R

21: function verifyPosConstraint(W, R, A)
22:    if {R' ∈ fulfilled[A] : R' ⊆ R} ≠ ∅ then
23:        return true
24:    if {R' ∈ unfulfilled[A] : R' ⊇ R} ≠ ∅ then
25:        return false
26:    result := ResProvable(W, R ⊃ A)
27:    if result then
28:        fulfilled[A] += R
29:    else
30:        unfulfilled[A] += R
31:    return result
```

The function uses the global maps fulfilled[] and unfulfilled[]. They map from a formula A to a set of sets of residues $\{R_1, \ldots, R_n\}$ and hold the information for which set of residues R_i the constraint $\mathbf{L}A$ is known to be fulfilled and not fulfilled in $\langle W, R_i \rangle$, respectively. Hence if $R \in$ fulfilled$[A]$ then $W, R \supset A$ is provable. If $R \in$ unfulfilled$[A]$ then $W, R \supset A$ is not provable.

The function first check whether R is a superset of a set R' for which we know that $\mathbf{L}A$ is fulfilled in $\langle W, R' \rangle$. If it is then we know that $\mathbf{L}A$ is also fulfilled in the given theory and return with success.

The function then checks whether R is a subset of a set R' for which we know that $\mathbf{L}A$ is not fulfilled in $\langle W, R' \rangle$. If it is then we know that $\mathbf{L}A$ is also not fulfilled in the given theory and return with failure.

If the previous two checks did not reveal any information the function verifies whether the constraint is fulfilled (line 26) and then caches the result accordingly (lines 27–30).

Improving Cached Results

We can narrow down the set of residues for which a constraint $\mathbf{L}A$ holds if we use in verifyPosConstraint a residue prover that provides us with a support $R_{\text{sprt}} \subseteq R$ of A for $\langle W, R \rangle$. That is instead of adding R to fulfilled$[A]$ we add the possibly smaller set R_{sprt} to it.

We can narrow down the set of residues in skippable$[\delta]$ accordingly. That is if $W, R \supset \neg B$ is valid for a justification $B \in \text{jus}(\delta)$ and $R_{\text{sprt}} \subseteq R$ is a support of it for $\langle W, R \rangle$ then we add R_{sprt} instead of R to skippable$[\delta]$.

Implementation Remarks

In our implementation we use vectors to store the defaults of a theory and to store the justifications of a default. We therefore identify a default or residue of a theory by its index and a justification of a default of a theory by a pair of indexes, i.e. the index of its default and its index in the vector of justifications. For the four global maps we use these indexes to identify the defaults and justifications. Hence skippable[] and nonskippable[] map from an index d, representing a default, to a set of indexes $\{r_1, \ldots, r_n\}$, representing a set of residues, and fulfilled[] and unfulfilled[] map from a pair of indexes $\langle d, j \rangle$, representing the j's justification of the d's default to a set of indexes $\{r_1, \ldots, r_n\}$, representing a set of residues.

With such maps the algorithm may be less effective because two equal justifications from different residues will have different map entries.

Furthermore two different defaults may have the same residual part. A set of residues may thus be a subset of another one while the corresponding sets of indexes are not in a subset relation.

Proof Search for Normal Default Theories

Reiter [24] shows that if a default theory is normal then an extension is known to exist. If a normal default theory has a consistent base theory then the extensions are known to be consistent, otherwise $\mathcal{L}$ is its only extension.

For proof search we take advantage of this fact. Let $\langle W, D\rangle$ be a normal default theory. To prove whether $\langle W, D\rangle$ credulously entails A we first verify whether $W \supset A$ is valid. If it is valid then we know — independent of whether the base theory is consistent or not — that A is in all extensions of $\langle W, D\rangle$ and can return with success.

If W does not entail A then W must be consistent. We then calculate the minimal requirements $\mathcal{D}_{\min}$ of A for $\langle W, D\rangle$. Now let $D_{\min} \in \mathcal{D}_{\min}$.

- If $W \cup \mathrm{con}(D_{\min})$ is consistent then it is easy to see that there exists an extension E with $D_{\min} \subseteq \mathrm{GD}(D, E)$.
- If $W \cup \mathrm{con}(D_{\min})$ is not consistent for $D_{\min} \in \mathcal{D}_{\min}$ then we know, because all extensions are consistent, that no extension E exists such that $\mathrm{GD}(D, E)$ contains $D_{\min}$.

We thus know that A is credulously entailed by $\langle W, D\rangle$ iff $W \cup \mathrm{con}(D_{\min})$ is consistent for some $D_{\min} \in \mathcal{D}_{\min}$.

7.2 Proof Search in Skeptical Default Logic

In this section we investigate backward proof search for skeptical default logic. We start with a simple prover that backward applies the rules of sPDC. As in the credulous case this prover has obvious redundancies and we show how to avoid them by reducing the branching grade. Then we introduce residual improvement for skeptical default logic. That is we show how to use the provability or refutability of residue sequents encountered in proof search to decide provability or refutability of other, yet unprocessed residue sequents encountered in proof search. We close the section with use-check for skeptical default logic.

7.2.1 A Simple Prover

A simple backward proof search algorithm for skeptical default logic can easily be given. Rule (sD4) is invertible while (sD1), (sD2) and (sD3) are not. Backtracking is thus only necessary for the rules that are defined on residue sequents.

Lemma 7.5 ((sD4) is invertible)
The rule (sD4) is invertible.

Proof. Because (sD4) is the only rule that deduces proper defaults, the claim follows directly from the soundness and completeness of sPDC. □

Remark 7.6 ((sD1), (sD2) and (sD3) are not invertible)
It is obvious that the rules (sD1), (sD2) and (sD3) are not invertible.

(sD1) Consider $\neg\mathbf{L}p; p \supset q$. This sequent is skeptically valid but $p \supset q$ is not valid.

(sD2) Consider $\neg\mathbf{L}q; p \supset p$. This sequent is skeptically valid but $p \supset q$ is not valid.

(sD3) Consider $\mathbf{L}p; p \supset p$. This sequent is skeptically valid but $p \supset p$ is valid, i.e. not refutable.

According to the previous lemma and remark we can give a simple algorithm for backward proof search in skeptical default logic (see Algorithm 21).

The algorithm consists of three functions.

skepDefProvable initiates the proof search. The given arguments are a default theory $\langle W, D\rangle$ and the formula A that is to prove.
The function first splits D into residues R and proper defaults D' (lines 2+3) and then starts proof search by calling sD4 (line 4).

sD4 applies rule (sD4). Its argument is a default sequent that has its defaults split into residues R and proper defaults D.
If D is empty then the function applies (sD1) (line 7) and if this is not successful calls sD23 to apply the rules (sD2) and (sD3).
If D is not empty then the function selects a proper default $\delta \in D$ as principal formula and recursively tries to prove the most left premise (line 13) and the other premises (lines 14–15).

sD23 applies rules (sD2) and (sD3). Its arguments is a default sequent that contains only residues.
The function simply checks whether for one of the given constraints the appropriate rule succeeds (lines 20–23).

Algorithm 21 Simple proof search for skeptical default logic

```
1: function skepDefProvable(⟨W, D⟩, A)
2:     R := {δ ∈ D : jus(δ) = ∅}
3:     D' := {δ ∈ D : jus(δ) ≠ ∅}
4:     return sD4(; W, R, D' ⊃ A)

5: function sD4(Σ; W, R, D ⊃ A)
6:     if D = ∅ then                ▷ only residues left, apply (sD1), (sD2), (sD3)
7:         result := resProvable(W, R ⊃ A)
8:         result := result or sD23(Σ; W, R ⊃ A)
9:     else                         ▷ have proper defaults, apply (sD4)
10:        choose δ ∈ D
11:        D' := D \ {δ}
12:        R' := R ∪ {pre(δ)/con(δ)}
13:        result := sD4(Σ ∪ {¬L¬B : B ∈ jus(δ)} ; W, R', D' ⊃ A)
14:        for B ∈ jus(δ) do
15:            result := result and sD4(Σ ∪ {L¬B} ; W, R, D' ⊃ A)
16:    return result

17: function sD23(Σ; W, R ⊃ A)
18:    result := false
19:    for σ ∈ Σ do
20:        if σ is of the form ¬L¬B then
21:            result := result or resProvable(W, R ⊃ ¬B)        ▷ (sD2)
22:        else
23:            result := result or resRefutable(W, R ⊃ ¬B)       ▷ (sD3)
24:    return result
```

$$;W, \frac{A_1 : B_{11}, B_{12}}{C_1}, \frac{A_2 : B_{21}, B_{22}}{C_2} \supset \Delta$$

$$0\ \neg\mathbf{L}\neg B_{11}, \neg\mathbf{L}\neg B_{12}; W, \frac{A_1}{C_1}, \frac{A_2 : B_{21}, B_{22}}{C_2} \supset \Delta$$

$$0.0\ \neg\mathbf{L}\neg B_{11}, \neg\mathbf{L}\neg B_{12}, \neg\mathbf{L}\neg B_{21}, \neg\mathbf{L}\neg B_{22}; W, \frac{A_1}{C_1}, \frac{A_2}{C_2} \supset \Delta$$

$$0.1\ \neg\mathbf{L}\neg B_{11}, \neg\mathbf{L}\neg B_{12}, \mathbf{L}\neg B_{21}; W, \frac{A_1}{C_1} \supset \Delta$$

$$0.2\ \neg\mathbf{L}\neg B_{11}, \neg\mathbf{L}\neg B_{12}, \mathbf{L}\neg B_{22}; W, \frac{A_1}{C_1} \supset \Delta$$

$$1\ \mathbf{L}\neg B_{11}; W, \frac{A_2 : B_{21}, B_{22}}{C_2} \supset \Delta$$

$$1.0\ \mathbf{L}\neg B_{11}, \neg\mathbf{L}\neg B_{21}, \neg\mathbf{L}\neg B_{22}; W, \frac{A_2}{C_2} \supset \Delta$$

$$1.1\ \mathbf{L}\neg B_{11}, \mathbf{L}\neg B_{21}; W \supset \Delta$$

$$1.2\ \mathbf{L}\neg B_{11}, \mathbf{L}\neg B_{22}; W \supset \Delta$$

$$2\ \mathbf{L}\neg B_{12}; W, \delta_2 \supset \Delta$$

$$2.0\ \mathbf{L}\neg B_{12}, \neg\mathbf{L}\neg B_{21}, \neg\mathbf{L}\neg B_{22}; W, \frac{A_2}{C_2} \supset \Delta$$

$$2.1\ \mathbf{L}\neg B_{12}, \mathbf{L}\neg B_{21}; W \supset \Delta$$

$$2.2\ \mathbf{L}\neg B_{12}, \mathbf{L}\neg B_{22}; W \supset \Delta$$

Figure 7.1: Redundancies in the simple prover

Avoiding Obvious Redundancies

The simple prover given above has redundancies. To illustrate this consider the residue sequents inspected by it for

$$\mathrm{S} =; W, \frac{A_1 : B_{11}, B_{12}}{C_1}, \frac{A_2 : B_{21}, B_{22}}{C_2} \supset \Delta.$$

Suppose that S is skeptically valid. Then the prover will encounter nine different sequents that contain only residues (cf. Figure 7.1). For each of them applying either (sD1), (sD2) or (sD3) will succeed.

Consider rule (sD1). This rule uses only the non-constraint part of the sequent. If we remove the constraints from the nine sequents in question then we obtain only four different sequents:

$$W, \frac{A_1}{C_1}, \frac{A_2}{C_2} \supset \Delta, \qquad W, \frac{A_1}{C_1} \supset \Delta, \qquad W, \frac{A_2}{C_2} \supset \Delta, \qquad W \supset \Delta.$$

Hence five of nine applications of (sD1) are redundant in this simple example.

Consider rule (sD2). It verifies for a negative constraint $\neg\mathbf{L}\neg B$ whether $\neg B$ is valid in the current antecedent. Consider the positions 0.1 and 0.2. If there applying (sD1) fails then the prover may check twice whether $W, A_1/C_1 \supset \neg B_{11}$ or $W, A_1/C_1 \supset \neg B_{12}$ is valid. At position 1.0 and 2.0 we have a similar situation for $\neg\mathbf{L}\neg B_{21}$ and $\neg\mathbf{L}\neg B_{22}$.

Consider rule (sD3). It verifies for a negative constraint $\mathbf{L}\neg B$ whether $\neg B$ is refutable in the current antecedent. The sequents at positions 1.1 and 1.2 contain the constraint $\mathbf{L}\neg B_{11}$. The prover may thus verify twice whether $W \supset \neg B_{11}$ is refutable. We have similar situations for $\mathbf{L}\neg B_{12}$ at positions 2.1 and 2.2, for $\mathbf{L}\neg B_{21}$ at positions 1.1 and 2.1 and for $\mathbf{L}\neg B_{22}$ at positions 1.2 and 2.2.

These redundancies arise if there are defaults with more than one justification. To avoid them we can delay branching over the justification until only residues are left. The idea is similar to the one we use in the credulous case where we partition the defaults into D_4 and D_5.

Instead of modifying the antecedent and the set of constraints for proof search we just split the defaults into two disjoint sets D_{in} and D_{out}. D_{in} holds the defaults whose residual part is to be added to the antecedent and reflects the most left premise of (sD4). D_{out} holds the defaults whose residual part is not to be added to the antecedent and reflects the other premises of (sD4).
From D_{in} we can calculate the residue theory in the antecedent and the negative constraints. It is obvious how to do this.
From D_{out} we can calculate the positive constraints. Now because a default $\delta \in D_{\text{out}}$ represents proving $|\text{jus}(\delta)|$ sequents, D_{out} reflects several sets of positive constraints. Namely those that contain from each default δ_i in D_{out} exactly one positive constraint $\mathbf{L}\neg B_i$ where B_i is a justification of δ_i. Such a set of positive constraints thus corresponds to a tuple of $\prod_{\delta\in D_{\text{out}}} \text{jus}(\delta)$.

To illustrate this approach consider our example in Figure 7.1. Let δ_1 be the first and δ_2 be the second default of our sequent. Then there are four possibilities to partition the defaults into $(D_{\text{in}}, D_{\text{out}})$.

$(\{\delta_1, \delta_2\}, \{\})$ reflects the set of sequents of position 0.0, i.e. those sequents having the non-constraint part $W, A_1/C_1, A_2/C_2 \supset \Delta$.

$(\{\delta_1\}, \{\delta_2\})$ reflects the set of sequents of position 0.1 and 0.2, i.e. those sequents having the non-constraint part $W, A_1/C_1 \supset \Delta$.

$(\{\delta_2\}, \{\delta_1\})$ reflects the set of sequents of position 1.0 and 2.0, i.e. those sequents having the non-constraint part $W, A_2/C_2 \supset \Delta$.

$(\{\}, \{\delta_1, \delta_2\})$ reflects the set of sequents of position 1.1, 1.2, 2.1 and 2.2, i.e.

	$(D_{\text{in}}, D_{\text{out}})$	Σ_{in}	$\mathbb{S}_{\text{out}}$	R_{in}
1	$(\{\delta_1, \delta_2\}, \{\})$	$\neg\mathbf{L}\neg B_{11}, \neg\mathbf{L}\neg B_{12}, \neg\mathbf{L}\neg B_{21}, \neg\mathbf{L}\neg B_{22}$	$\emptyset$	$\frac{A_1}{C_1}, \frac{A_2}{C_2}$
2	$(\{\delta_1\}, \{\delta_2\})$	$\neg\mathbf{L}\neg B_{11}, \neg\mathbf{L}\neg B_{12}$	$\{\mathbf{L}\neg B_{21}\}, \{\mathbf{L}\neg B_{22}\}$	$\frac{A_1}{C_1}$
3	$(\{\delta_2\}, \{\delta_1\})$	$\neg\mathbf{L}\neg B_{21}, \neg\mathbf{L}\neg B_{22}$	$\{\mathbf{L}\neg B_{11}\}, \{\mathbf{L}\neg B_{12}\}$	$\frac{A_2}{C_2}$
4	$(\{\}, \{\delta_1, \delta_2\})$	$\emptyset$	$\{\mathbf{L}\neg B_{11}, \mathbf{L}\neg B_{21}\}, \{\mathbf{L}\neg B_{11}, \mathbf{L}\neg B_{22}\}, \{\mathbf{L}\neg B_{12}, \mathbf{L}\neg B_{21}\}, \{\mathbf{L}\neg B_{12}, \mathbf{L}\neg B_{22}\}$	$\emptyset$

Table 7.1: Sequents encountered when partitioning the defaults

those sequents having the non-constraint part $W \supset \Delta$.

The general case is covered by the following definition.

Definition 7.7 ($\mathcal{S}_{(D_{\mathbf{in}}, D_{\mathbf{out}})}$)
Let $; W, D \supset \Delta$ be the default sequent and $(D_{\text{in}}, D_{\text{out}})$ a disjoint partition of D with $D_{\text{out}} = \{\delta_1, \delta_2, \ldots, \delta_m\}$. Furthermore let

$$R_{\text{in}} = \left\{ \frac{\text{pre}(\delta)}{\text{con}(\delta)} : \delta \in D_{\text{in}} \right\}$$
$$\Sigma_{\text{in}} = \{\neg\mathbf{L}\neg\text{pre}(\delta) : \delta \in D_{\text{in}}\}$$
$$\mathbb{S}_{\text{out}} = \left\{ \{\mathbf{L}\neg B_1, \mathbf{L}\neg B_2, \ldots, \mathbf{L}\neg B_m\} : \langle B_1, B_2, \ldots, B_m\rangle \in \prod_{i=1}^{m} \text{jus}(\delta_i) \right\}.$$

Then we define the set $\mathcal{S}_{(D_{in}, D_{out})}$ of default sequents as

$$\mathcal{S}_{(D_{\text{in}}, D_{\text{out}})} = \{\Sigma_{\text{in}}, \Sigma_{\text{out}}; W, R_{\text{in}} \supset \Delta : \Sigma_{\text{out}} \in \mathbb{S}_{\text{out}}\}.$$

For an example consider Table 7.1 which lists the sets Σ_{in}, $\mathbb{S}_{\text{out}}$ and R_{in} of our example according to the possible partitions.

The procedure to verify whether the sequents in a set $\mathcal{S}_{(D_{\text{in}}, D_{\text{out}})}$ are all skeptically valid is straight forward.

1. Check whether $W, R_{\text{in}} \supset \Delta$ is valid. This corresponds to the backward application of (sD1) for all sequents in $\mathcal{S}_{(D_{\text{in}}, D_{\text{out}})}$.
2. If the above check did not succeed then check whether $W, R_{\text{in}} \supset \neg B$ is valid for some $\neg\mathbf{L}\neg B \in \Sigma_{\text{in}}$. This corresponds to the backward application of (sD2) for all sequents in $\mathcal{S}_{(D_{\text{in}}, D_{\text{out}})}$.

3. If the above two checks did not succeed then check whether every $\Sigma_{\text{out}} \in \mathbb{S}_{\text{out}}$ contains a constraint $\mathbf{L}\neg B$ such that $W, R_{\text{in}} \supset \neg B$ is refutable. This corresponds to the backward application of (sD3) for all sequents in $\mathcal{S}_{(D_{\text{in}}, D_{\text{out}})}$.

The last point given above is equivalent to check whether $W, R_{\text{in}} \supset \neg B_i$ is refutable for all $B_i \in \text{jus}(\delta)$ of an arbitrary default $\delta \in D_{\text{out}}$. This equivalence is easy to see.
Suppose that there is no default $\delta \in D_{\text{out}}$ for which this is fulfilled. Then there exists for each default $\delta_i \in D_{\text{out}}$ a justification $B_i \in \text{jus}(\delta_i)$ for which $W, R_{\text{in}} \supset \neg B_i$ is not refutable. Hence there is a tuple $\langle B_1, \ldots, B_m \rangle \in \prod_{i=1}^{m} \text{jus}(\delta_i)$ such that $W, R_{\text{in}} \supset \neg B_i$ is not refutable for $1 \leq i \leq m$.
Suppose that there is a default $\delta_j \in D_{\text{out}}$ for which this is fulfilled. Since every tuple $\langle B_1, B_2, \ldots, B_m \rangle \in \prod_{i=1}^{m} \text{jus}(\delta_i)$ contains a justification of δ_j we then know that every $\Sigma_{\text{out}} \in \mathbb{S}_{\text{out}}$ contains a constraint $\mathbf{L}\neg B_j$ such that $W, R_{\text{in}} \supset \neg B_j$ is refutable.

The algorithm for this approach is given in Algorithm 22 and consists of three function.

skepDefProvable' initiates the proof search. The given arguments are a default theory $\langle W, D \rangle$ and the formula A that is to prove.
The function first moves the residues of D to D_{in} and the proper residues to D' (lines 2+3) and starts proof search by calling sD4 (line 4).

sD4 recursively creates all possible disjoint partitions of the set of defaults and calls sD123 to verify a partition. The given argument are two sets of defaults D_{in} and D_{out} representing a partial partition of the defaults, the base theory W, the set of defaults D that is still to partition and the formula A that is to prover.

sD123 checks whether the sequents represented by the given partition are skeptically valid. Its arguments are a disjoint partition $(D_{\text{in}}, D_{\text{out}})$ of the defaults, the base theory W and the formula A that is to prove.
The function proceeds according to the three steps given above. It first verifies whether $W, R_{\text{in}} \supset A$ is valid (line 15). If this fails it checks the negative constraints (lines 16–18). If this also fails then it checks the positive constraints (lines 19–24).

A further advantage of that approach is that we use in sD123 the same residue theory to prove or refute different formulas. If we there use the prover based on minimal quasi-supports, then the cached information regarding minimal quasi-supports of each residue can be reused. If we use the prover based on the closure then $\text{Cl}'(W, R)$ needs to be calculated only to once.

Algorithm 22 Partition based proof search for skeptical default logic

```
1: function skepDefProvable'(⟨W, D⟩, A)
2:     D' := {δ ∈ D : jus(δ) ≠ ∅}
3:     D_in := {δ ∈ D : jus(δ) = ∅}
4:     return sD4(D_in, ∅, W, D', A)

5: function sD4(D_in, D_out, W, D, A)
6:     if D = ∅ then
7:         result := sD123(D_in, D_out, W, A)
8:     else
9:         choose δ ∈ D
10:        result := sD4(D_in ∪ {δ}, D_out, W, D \ {δ}, A) and
11:                  sD4(D_in, D_out ∪ {δ}, W, D \ {δ}, A)
12:    return result

13: function sD123(D_in, D_out, W, A)
14:    R_in := {pre(δ)/con(δ) : δ ∈ D_in}
15:    result := resProvable(W, R_in ⊃ A)                    ▷ try (sD1)
16:    if not result then          ▷ (sD1) not successful, try (sD2)
17:        for B ∈ ⋃_{δ∈D_in} jus(δ) do
18:            result := result or resProvable(W, R_in ⊃ ¬B)
19:    if not result then      ▷ (sD1) and not (sD2) successful, try (sD3)
20:        for δ ∈ D_out do
21:            result := true
22:            for B ∈ jus(δ) do
23:                result := result and resRefutable(W, R_in ⊃ ¬B)
24:            if result then break
25:    return result
```

7.2.2 Residual Improvement

As in the credulous case there are also situations where we can conclude the provability or refutability of a premise of (sD1), (sD2) or (sD3) from previously gained results, because the fragment of default logic reduced to residues is monotonic. To illustrate this consider our example in Table 7.1.

1.0 If (sD1) fails on line 2, i.e. $W, A_1/C_1 \supset \Delta$ is not provable, then (sD1) is known to fail on line 4, i.e. $W \supset \Delta$ is not provable.

1.1 If (sD1) succeeds on line 2, i.e. $W, A_1/C_1 \supset \Delta$ is provable, then (sD1) is known to succeed on line 1, i.e. $W, A_1/C_1, A_2/C_2 \supset \Delta$ is provable.

2.0 If (sD2) fails for $\neg\mathbf{L}\neg B_{11}$ on line 1, i.e. $W, A_1/C_1, A_2/C_2 \supset \neg B_{11}$ is not provable, then (sD2) is known to fail for $\neg\mathbf{L}\neg B_{11}$ on line 2, i.e. $W, A_1/C_1 \supset \neg B_{11}$ is not provable,.

2.1 If (sD2) succeeds for $\neg\mathbf{L}\neg B_{11}$ on line 2, i.e. $W, A_1/C_1 \supset \neg B_{11}$ is provable, then (sD2) is known to succeed for $\neg\mathbf{L}\neg B_{11}$ on line 1, i.e. $W, A_1/C_1, A_2/C_2 \supset \neg B_{11}$ is provable.

3.0 If (sD3) fails for $\mathbf{L}\neg B_{11}$ on line 4, i.e. $W \supset \neg B_{11}$ is not refutable, then (sD3) is known to fail for $\mathbf{L}\neg B_{11}$ on line 3, i.e. $W, A_2/C_2 \supset \neg B_{11}$ is not refutable.

3.1 If (sD3) succeeds for $\mathbf{L}\neg B_{11}$ on line 3, i.e. $W, A_2/C_2 \supset \neg B_{11}$ is refutable, then (sD3) is known to succeed for $\mathbf{L}\neg B_{11}$ on line 4, i.e. $W \supset \neg B_{11}$ is refutable.

The general scheme of this easy to see. Each line in our figure represents a disjoint partition of the defaults D of our default theory. Let $(D_{\text{in}}, D_{\text{out}})$ and $(D'_{\text{in}}, D'_{\text{out}})$ be two such partitions and R_{in} and R'_{in} as in definition 7.7. We use the fact that if $D_{\text{in}} \subseteq D'_{\text{in}}$ then $R_{\text{in}} \subseteq R'_{\text{in}}$.

1.0 If (sD1) fails for partition $(D_{\text{in}}, D_{\text{out}})$ then it will fail for any partition $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \subseteq D_{\text{in}}$, because

$$\mathsf{PRC} \nvdash W, R_{\text{in}} \supset \Delta \quad \Longrightarrow \quad \mathsf{PRC} \nvdash W, R'_{\text{in}} \supset \Delta \quad \text{if} \quad R'_{\text{in}} \subseteq R_{\text{in}}.$$

1.1 If (sD1) succeeds for partition $(D_{\text{in}}, D_{\text{out}})$ then it will succeed for any partition $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \supseteq D_{\text{in}}$, because

$$\mathsf{PRC} \vdash W, R_{\text{in}} \supset \Delta \quad \Longrightarrow \quad \mathsf{PRC} \vdash W, R'_{\text{in}} \supset \Delta \quad \text{if} \quad R'_{\text{in}} \supseteq R_{\text{in}}.$$

2.0 If (sD2) fails for constraint $\neg\mathbf{L}\neg B$ for partition $(D_{\text{in}}, D_{\text{out}})$ then it will fail for $\neg\mathbf{L}\neg B$ for any partition $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \subseteq D_{\text{in}}$, because

$$\mathsf{PRC} \nvdash W, R_{\text{in}} \supset \neg B \quad \Longrightarrow \quad \mathsf{PRC} \nvdash W, R'_{\text{in}} \supset \neg B \quad \text{if} \quad R'_{\text{in}} \subseteq R_{\text{in}}.$$

2.1 If (sD2) succeeds for constraint $\neg\mathbf{L}\neg B$ for partition $(D_{\text{in}}, D_{\text{out}})$ then it will succeed for $\neg\mathbf{L}\neg B$ for any partition $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \supseteq D_{\text{in}}$, because

$$\mathsf{PRC} \vdash W, R_{\text{in}} \supset \neg B \quad \Longrightarrow \quad \mathsf{PRC} \vdash W, R'_{\text{in}} \supset \neg B \quad \text{if} \quad R'_{\text{in}} \supseteq R_{\text{in}}.$$

3.0 If (sD3) fails for constraint $\mathbf{L}\neg B$ for partition $(D_{\text{in}}, D_{\text{out}})$ then it will fail for $\mathbf{L}\neg B$ for any partition $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \supseteq D_{\text{in}}$, because

$$\mathsf{PRRC} \nvdash W, R_{\text{in}} \supset \neg B \quad \Longrightarrow \quad \mathsf{PRRC} \nvdash W, R'_{\text{in}} \supset \neg B \quad \text{if} \quad R'_{\text{in}} \supseteq R_{\text{in}}.$$

3.1 If (sD3) succeeds for constraint $\mathbf{L}\neg B$ for partition $(D_{\text{in}}, D_{\text{out}})$ then it will succeed for $\mathbf{L}\neg B$ for any partition $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \subseteq D_{\text{in}}$, because

$$\mathsf{PRRC} \vdash W, R_{\text{in}} \supset \neg B \quad \Longrightarrow \quad \mathsf{PRRC} \vdash W, R'_{\text{in}} \supset \neg B \quad \text{if} \quad R'_{\text{in}} \subseteq R_{\text{in}}.$$

Lifting Residual Improvement

The general scheme of the rules (sD2) and (sD3) applies to constraints. We can lift it from the constraint to the partition level.

Let $\langle W, D\rangle$ be a default theory, $(D_{\text{in}}, D_{\text{out}})$ and $(D'_{\text{in}}, D'_{\text{out}})$ be two disjoint partitions of D and R_{in}, Σ_{in}, R'_{in} and Σ'_{in} be as in definition 7.7.

2.0 Suppose that (sD2) fails for all constraints in Σ_{in}, i.e.

$$\mathsf{PRC} \nvdash W, R_{\text{in}} \supset \neg B \text{ for all } B \in \bigcup_{\delta \in D_{\text{in}}} \text{jus}(\delta).$$

Let $R'_{\text{in}} \subseteq R_{\text{in}}$. Then we know

$$\mathsf{PRC} \nvdash W, R'_{\text{in}} \supset \neg B \text{ for all } B \in \bigcup_{\delta \in D_{\text{in}}} \text{jus}(\delta).$$

$R'_{\text{in}} \subseteq R_{\text{in}}$ implies $D'_{\text{in}} \subseteq D_{\text{in}}$, we thus know

$$\mathsf{PRC} \nvdash W, R'_{\text{in}} \supset \neg B \text{ for all } B \in \bigcup_{\delta' \in D'_{\text{in}}} \text{jus}(\delta').$$

Hence (sD2) is known to fail for all constraints in Σ'_{in} if $D'_{\text{in}} \subseteq D_{\text{in}}$.

2.1 Suppose that (sD2) succeeds for a constraint $\neg\mathbf{L}\neg B \in \Sigma_{\text{in}}$, i.e.

$$\mathsf{PRC} \vdash W, R_{\text{in}} \supset \neg B.$$

Let $R'_{\text{in}} \supseteq R_{\text{in}}$. Then we know $\mathsf{PRC} \vdash W, R'_{\text{in}} \supset \neg B$.

$R_{\text{in}} \subseteq R'_{\text{in}}$ implies $\Sigma_{\text{in}} \subseteq \Sigma'_{\text{in}}$, therefore our witness $\neg\mathbf{L}\neg B$ is also in Σ'_{in}. Hence (sD2) is known to succeed for $\neg\mathbf{L}\neg B \in \Sigma'_{\text{in}}$ if $D_{\text{in}} \subseteq D'_{\text{in}}$.

3.0 Suppose that for every $\delta \in D_{\text{out}}$ there exists $B \in \text{jus}(\delta)$ such that

$$\mathsf{PRRC} \nvdash W, R_{\text{in}} \supset \neg B.$$

Let $R'_{\text{in}} \supseteq R_{\text{in}}$. Then we know $\mathsf{PRRC} \nvdash W, R'_{\text{in}} \supset \neg B$ (1).

$R_{\text{in}} \subseteq R'_{\text{in}}$ implies $D'_{\text{out}} \subseteq D_{\text{out}}$. Hence we know that for every $\delta' \in D'_{\text{out}}$ there exists $B \in \text{jus}(\delta')$ such that (1) holds.

3.1 Suppose that there exists $\delta \in D_{\text{out}}$ such that

$$\mathsf{PRRC} \vdash W, R_{\text{in}} \supset \neg B \text{ for all } B \in \text{jus}(\delta).$$

Let $R'_{\text{in}} \subseteq R_{\text{in}}$. Then we know

$$\mathsf{PRRC} \vdash W, R'_{\text{in}} \supset \neg B \text{ for all } B \in \text{jus}(\delta).$$

$R'_{\text{in}} \subseteq R_{\text{in}}$ implies $D_{\text{out}} \subseteq D'_{\text{out}}$, hence our witness δ is also in D'_{out}.

The lifted the general scheme of rule (sD2) turns out to be equivalent to the scheme of (sD1). We therefore end up with four different cases.

1. If (sD1) and (sD2) fail for $(D_{\text{in}}, D_{\text{out}})$ then they also fail for $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \subseteq D_{\text{in}}$.
2. If (sD1) or (sD2) succeeds for $(D_{\text{in}}, D_{\text{out}})$ then the corresponding rule also succeeds for $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \supseteq D_{\text{in}}$.
3. If (sD3) fails for $(D_{\text{in}}, D_{\text{out}})$ then it also fails for $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \supseteq D_{\text{in}}$.
4. If (sD3) succeeds for $(D_{\text{in}}, D_{\text{out}})$ then it also succeeds for $(D'_{\text{in}}, D'_{\text{out}})$ with $D'_{\text{in}} \subseteq D_{\text{in}}$.

To make use of the scheme we need to remember whether a partition failed for (sD1) and (sD2), succeeded for (sD1) or (sD2), failed for (sD3), or succeeded for (sD3). Then we use that information to prevent superfluous provability checks.

A modified version of the function sD123 is given in Algorithm 23. It uses the global variables $\mathcal{R}_{12}^{\perp}$, $\mathcal{R}_{12}^{\top}$, $\mathcal{R}_{3}^{\perp}$ and $\mathcal{R}_{3}^{\top}$. They hold those sets of residues R_{in} for which (sD1) and (sD2) failed, (sD1) or (sD2) succeeded, (sD3) failed and (sD3) succeeded, respectively.
There are two differences to sD123 of Algorithm 22.

1. It first checks whether the result for the given partition $(D_{\text{in}}, D_{\text{out}})$ is already known (lines 3–6). If yes, it immediately returns with the appropriate result, otherwise it proceeds as in Algorithm 22.
2. It updates the four global variables according to the result (lines 11–14 and 21–24)

Algorithm 23 Residual improvement in proof search for skeptical default logic

```
1: function sD123R(D_in, D_out, W, A)
2:     R_in := {pre(δ)/con(δ) : δ ∈ D_in}
3:     if {R ∈ R_12^⊤ : R ⊆ R_in} ≠ ∅ or {R ∈ R_3^⊤ : R ⊇ R_in} ≠ ∅ then
4:         return true                                   ▷ known to succeed
5:     if {R ∈ R_12^⊥ : R ⊇ R_in} ≠ ∅ or {R ∈ R_3^⊥ : R ⊆ R_in} ≠ ∅ then
6:         return false                                  ▷ known to fail
7:     result := resProvable(W, R_in ⊃ A)                ▷ try (sD1)
8:     if not result then                  ▷ (sD1) not successful, try (sD2)
9:         for B ∈ ⋃_{δ∈D_in} jus(δ) do
10:            result := result or resProvable(W, R_in ⊃ ¬B)
11:    if result then
12:        R_12^⊤ := R_12^⊤ ∪ {R_in}          ▷ remember success of (sD1)/(sD2)
13:    else
14:        R_12^⊥ := R_12^⊥ ∪ {R_in}          ▷ remember failure of (sD1)/(sD2)
15:    if not result then      ▷ (sD1) and not (sD2) successful, try (sD3)
16:        for δ ∈ D_out do
17:            result := true
18:            for B ∈ jus(δ) do
19:                result := result and resRefutable(W, R_in ⊃ ¬B)
20:            if result then break
21:        if result then
22:            R_3^⊤ := R_3^⊤ ∪ {R_in}            ▷ remember success of (sD3)
23:        else
24:            R_3^⊥ := R_3^⊥ ∪ {R_in}            ▷ remember failure of (sD3)
25:    return result
```

Improving Cached Results

As in the credulous case we can narrow down the set of residues R_{in} for which (sD1) or (sD2) holds if we use a residue prover in function sD123R that provides us with a support R_{sprt} of the proven sequent (lines 7 or 10). That is we add the possibly smaller set R_{sprt} instead of R_{in} to $\mathcal{R}_{12}^{\top}$.

We can also use the supports to narrow down the set or residues stored in $\mathcal{R}_{3}^{\perp}$. However, this case is more complicated because there we need for each default $\delta_i \in D_{\text{out}}$ a justification $B_i \in \delta_i$ for which $W, R_{\text{in}} \supset \neg B_i$ is not refutable, i.e. for which that sequent is provable. We hence have $|D_{\text{out}}|$ supports: For each such justification B_i a support $R_{\text{sprt,i}}$. The set that we have to store in $\mathcal{R}_{3}^{\perp}$ instead of R_{in} is thus the union of those supports $R_{\text{sprt,i}}$. By uniting the supports $R_{\text{sprt,i}}$ we loose detailed information about the failed refutations. It is therefore advantageous to also maintain for each default $\delta \in \delta$ a set $\mathcal{R}_{\delta 3}^{\perp}$ and to consult that set to check whether the refutation of a constraint derived from δ is know to fail before executing the loop on lines 18–19.

Implementation Remarks

As in the credulous case our implementation differs from the pseudocode. In the implementation we use vectors to store the defaults of a theory and the justifications of a default. We therefore identify a default and its residual part by its index and the justification of a default of a theory by a pair of indexes. For $\mathcal{R}_{12}^{\perp}$, $\mathcal{R}_{12}^{\top}$, $\mathcal{R}_{3}^{\perp}$ and $\mathcal{R}_{3}^{\top}$ we use these indexes instead of residues and may thus become less effective because two different defaults may have the same residual part. A set of residues may thus be a subset of another one while the corresponding sets of indexes are not in a subset relation.

7.2.3 Use-Check

In this section we show a method to avoid superfluous calls to sD4, namely the second recursive call on line 11. To successfully apply the method we need in sD123 a residue prover that provides us on success with the corresponding support. The idea behind use-check is as follows.

Examine the recursion in sD4. In the first recursive call (line 10) we extend the given partial partition $(D_{\text{in}}, D_{\text{out}})$ to $(D_{\text{in}} \cup \{\delta\}, D_{\text{out}})$. Hence every successive call of sD123 processes a partition $(D'_{\text{in}} \cup \{\delta\}, D'_{\text{out}})$ with $D_{\text{in}} \subseteq D'_{\text{in}}$ and $D_{\text{out}} \subseteq D'_{\text{out}}$.
Suppose that the first recursive call succeeds. Then we know that sD123 succeeds for every partition $(D'_{\text{in}} \cup \{\delta\}, D'_{\text{out}})$ with $D_{\text{in}} \subseteq D'_{\text{in}}$ and $D_{\text{out}} \subseteq$

D'_{out}. Then we processes the second recursive call (line 11). There we extend $(D_{\text{in}}, D_{\text{out}})$ to $(D_{\text{in}}, D_{\text{out}} \cup \{\delta\})$. Hence every successive call of sD123 processes a partition $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ with $D_{\text{in}} \subseteq D'_{\text{in}}$ and $D_{\text{out}} \subseteq D'_{\text{out}}$.

For every $(D'_{\text{in}} \cup \{\delta\}, D'_{\text{out}})$ in the first recursive call there exists thus a counterpart $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ in the second recursive call. Under certain circumstances we can deduce the success of $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ from the success of $(D'_{\text{in}} \cup \{\delta\}, D'_{\text{out}})$. These circumstances depend on the succeeding rule.

(sD1) Suppose that (sD1) succeeds and the support that witnesses the success does not contain the residue derived by δ, i.e. the witness is a subset of $\{\text{pre}(\delta)/\text{con}(\delta) : \delta \in D'_{\text{in}}\}$.
Then we know that (sD1) will also succeed with the same witness for $(D'_{\text{in}}, D'_{\text{out}})$ and hence also for $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$.

(sD2) Suppose that (sD2) succeeds and the support that witnesses the success does not contain the residue derived by δ, i.e. the witness is a subset of $\{\text{pre}(\delta)/\text{con}(\delta) : \delta \in D'_{\text{in}}\}$. Furthermore suppose that the negative constraint $\neg\mathbf{L}\neg B$ for which (sD2) succeeds does not derive from a justification of δ, i.e. $B \in \text{jus}(D'_{\text{in}})$.
Then we know that (sD2) also succeeds with the same witness for $(D'_{\text{in}}, D'_{\text{out}})$ and hence also for $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ for the same negative constraint $\neg\mathbf{L}\neg B$.

(sD3) Suppose that (sD3) succeeds for $(D'_{\text{in}} \cup \{\delta\}, D'_{\text{out}})$.
Then it is known to also succeed for $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ according to residual improvement.

Consider the calls of sD123 that succeed in the first recursive call. If in each of these calls we encounter one of the above cases then we know that the second recursive call will also be successful and can thus omit it.

According to the cases above the way to apply use-check is obvious. We mark those defaults as being used whose corresponding residues appear in the witness of successful (sD1) and (sD2) applications. If (sD2) succeeds for a negative constraint $\neg\mathbf{L}\neg B$ then we also mark that default as being used from which the constraint derives.

Example

An example for use-check is given in Figure 7.2. It shows the encountered calls to resProvable and resRefutable together with the resulting set of used defaults for the sequent

$$p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$$

$p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$

1 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q$

1.1 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$

1.1.1 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$

(sD1): $\mathsf{PRC} \vdash p, q, \frac{p}{p_1}, \frac{q}{q_1}, \frac{q_1}{q_2} \supset q_2$

used[1.1.1] $= \left\{ \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \right\}$

1.1.2 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$

(sD1): $\mathsf{PRC} \nvdash p, q, \frac{p}{p_1}, \frac{q}{q_1} \supset q_2$

(sD2): $\mathsf{PRC} \nvdash p, q, \frac{p}{p_1}, \frac{q}{q_1} \supset \neg p_1$, $\mathsf{PRC} \nvdash p, q, \frac{p}{p_1}, \frac{q}{q_1} \supset \neg q_1$

(sD3): $\mathsf{PRRC} \vdash p, q, \frac{p}{p_1}, \frac{q}{q_1} \supset \neg q_2$

used[1.1.2] $= \{\}$

used[1.1] $=$ used[1.1.1] $\cup$ used[1.1.2] $= \left\{ \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \right\}$

1.2 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$

1.2.1 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$

(sD1): $\mathsf{PRC} \nvdash p, q, \frac{p}{p_1}, \frac{q_1}{q_2} \supset q_2$

(sD2): $\mathsf{PRC} \nvdash p, q, \frac{p}{p_1}, \frac{q_1}{q_2} \supset \neg p_1$, $\mathsf{PRC} \nvdash p, q, \frac{p}{p_1}, \frac{q_1}{q_2} \supset \neg q_2$

(sD3): $\mathsf{PRRC} \vdash p, q, \frac{p}{p_1}, \frac{q_1}{q_2} \supset \neg q_1$

used[1.2.1] $= \{\}$

1.2.2 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q_2$ can be omitted.

used[1.2] $=$ used[1.2.1] $= \{\}$

used[1] $=$ used[1.1] $\cup$ used[1.2] $= \left\{ \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \right\}$

2 $p, q, \frac{p : p_1}{p_1}, \frac{q : q_1}{q_1}, \frac{q_1 : q_2}{q_2} \supset q$ can be omitted.

Figure 7.2: Use-check in sPDC
Elements of D_{in} are marked green, those of D_{out} are marked red.

At 1.1.1 (sD1) succeeds with support $\{q/q_1, q_1/q_2\}$. We remember the corresponding defaults in used[1.1.1].
Since $q_1 : q_2/q_2$ is marked as being used we have to process 1.1.2. There (sD3) succeeds. The set used[1.1.2] of used defaults is thus empty.
Having processed both branches of 1.1 we can compute its set of used defaults used[1.1] = used[1.1.1] ∪ used[1.1.2]. Because $q : q_1/q_1$ is in used[1.1] we have to process 1.2.
At 1.2.1 (sD3) succeeds. The set used[1.2.1] of used defaults is thus empty. Because $q_1 : q_2/q_2$ is not marked as being used in used[1.2.1] we can omit proving 1.2.2 and know that there exists a proof of it with used[1.2.1] as set of used defaults.
Because we have not processed 1.2.2 the set of used defaults of 1.2 is simply the one of 1.2.1.
From used[1.1] and used[1.2] we can now compute the set used[1] of used defaults for 1. Because there $p : p_1/p_1$ is not marked as being used we can omit proving 2 and can return with success.

Switching Recursive Calls

Use-check is also applicable if we switch the recursive calls in sD4 of Algorithm 22, that is if we call sD4(D_{in}, $D_{\text{out}} \cup \{\delta\}$, W, $D \setminus \{\delta\}$, A) before sD4($D_{\text{in}} \cup \{\delta\}$, D_{out}, W, $D \setminus \{\delta\}$, A).

Consider a partition $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ and its counterpart $(D'_{\text{in}} \cup \{\delta\}, D_{\text{out}})$. We then encounter the following situations.

1. If (sD1) or (sD2) is successful for $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ then we know from residual improvement that the corresponding rule will also be successful for $(D'_{\text{in}} \cup \{\delta\}, D'_{\text{out}})$.

2. If (sD3) is successful for $(D'_{\text{in}}, D'_{\text{out}} \cup \{\delta\})$ then it is unknown whether (sD3) is successful for $(D'_{\text{in}} \cup \{\delta\}, D'_{\text{out}})$.

We can thus omit the second recursive call if in the first recursive call rule (sD3) never succeeded.

Verifying CPC Provability First

In the skeptical case there is no special procedure for normal default theories. We can however take up the idea to first verify CPC provability. If A follows from the base theory W of a default theory $\langle W, D\rangle$, then A is known to be in every extension of $\langle W, D\rangle$ because every extension includes W.

7.3 Extension Based Proving

A further approach to prove default sequents is to use extension based proving, i.e. to calculate the extensions of a default theory and to check whether the succedent is in one extension (credulous case) or in all extensions (skeptical case). To have a comparison to the sequent calculus based provers we have implemented a simple algorithm to compute the extensions of a default theory $\langle W, D\rangle$. The algorithm we use is a refinement of the "naive" method introduced by Marek and Truszczyński [19] who compute extensions based on well-orderings of the defaults.

Instead of well-orderings we use so called default-chains to compute extensions and start by introducing some definitions we need for our algorithm.

Definition 7.8 (applicable default)
Let T be a set of formulas. A default δ is called *applicable* for T if

$$T \vdash \mathrm{pre}(\delta) \quad \text{and} \quad T \nvdash \neg B \text{ for all } B \in \mathrm{jus}(\delta).$$

Definition 7.9 (default-chain)
Let $\langle W, D\rangle$ be a finite default theory and $\mathbf{D} := (\delta_1, \ldots, \delta_n)$ be a sequence of pairwise distinct defaults of D. Then $\mathbf{D}$ is called a *default-chain* of $\langle W, D\rangle$ if

1. δ_i is applicable for $W \cup \{\mathrm{con}(\delta_1), \ldots, \mathrm{con}(\delta_{i-1})\}$ for all $\delta_i \in \mathbf{D}$,

2. $W, \mathrm{con}(\mathbf{D}) \nvdash \neg B$ for all $B \in \mathrm{jus}(\mathbf{D})$.

A default-chain $\mathbf{D} = (\delta_1, \ldots, \delta_n)$ of $\langle W, D\rangle$ is called *extendable* if there exists $\delta \in D \setminus \mathbf{D}$ such that $(\delta_1, \ldots, \delta_n, \delta)$ is a default-chain of $\langle W, D\rangle$. We then call δ *appendable* to $\mathbf{D}$ for W and write $\mathbf{D} \circ \delta$ for the extended default-chain.

A default-chain is called *maximal* if it is not extendable.

Two default-chains that contain the same elements are called *set-equal*.

Notation 7.10 (default-chain as set of defaults)
As in the definition above we often write $\mathbf{D}$ to refer the set containing the elements of $\mathbf{D}$, for example $\delta \in \mathbf{D}$, $\mathrm{con}(\mathbf{D})$ or $\mathrm{jus}(\mathbf{D})$. It always follows from the context whether we refer to a default-chain or to the set of its elements.

Example 7.11 (default-chains)
Let $W := \emptyset$ and $D := \left\{ \frac{:q}{q}, \frac{:r}{r}, \frac{r:s}{s}, \frac{r:\neg s}{\neg s}, \frac{:\neg q, \neg r}{t} \right\}$.

Then the following default-chains of $\langle W, D\rangle$ exist:

$$\mathbf{D}_\emptyset := ()$$
$$\mathbf{D}_1 := \left(\frac{:q}{q}\right) \quad \mathbf{D}_2 := \left(\frac{:r}{r}\right) \quad \mathbf{D}_5 := \left(\frac{:\neg q, \neg r}{t}\right)$$
$$\mathbf{D}_{12} := \left(\frac{:q}{q}, \frac{:r}{r}\right) \quad \mathbf{D}_{21} := \left(\frac{:r}{r}, \frac{:q}{q}\right)$$
$$\mathbf{D}_{23} := \left(\frac{:r}{r}, \frac{r:s}{s}\right) \quad \mathbf{D}_{24} := \left(\frac{:r}{r}, \frac{r:\neg s}{\neg s}\right)$$
$$\mathbf{D}_{123} := \left(\frac{:q}{q}, \frac{:r}{r}, \frac{r:s}{s}\right) \quad \mathbf{D}_{124} := \left(\frac{:q}{q}, \frac{:r}{r}, \frac{r:\neg s}{\neg s}\right)$$
$$\mathbf{D}_{213} := \left(\frac{:r}{r}, \frac{:q}{q}, \frac{r:s}{s}\right) \quad \mathbf{D}_{214} := \left(\frac{:r}{r}, \frac{:q}{q}, \frac{r:\neg s}{\neg s}\right)$$
$$\mathbf{D}_{231} := \left(\frac{:r}{r}, \frac{r:s}{s}, \frac{:q}{q}\right) \quad \mathbf{D}_{241} := \left(\frac{:r}{r}, \frac{r:\neg s}{\neg s}, \frac{:q}{q}\right).$$

The default-chains $\mathbf{D}_5$, $\mathbf{D}_{123}$, $\mathbf{D}_{124}$, $\mathbf{D}_{213}$, $\mathbf{D}_{214}$, $\mathbf{D}_{231}$ and $\mathbf{D}_{241}$ are maximal for $\langle W, D\rangle$, the others are extendable.
The example also shows that two different default-chains can have the same elements. $\mathbf{D}_{123}$ and $\mathbf{D}_{231}$ for example are set-equal.

The following two sequences are no default-chains of $\langle W, D\rangle$:

$$\mathbf{D}_{32} := \left(\frac{r:s}{s}, \frac{:r}{r},\right) \qquad \mathbf{D}_{51} := \left(\frac{:\neg q, \neg r}{t}, \frac{:q}{q}\right)$$

Although $\mathbf{D}_{32}$ contains the same defaults as $\mathbf{D}_{23}$ it is not a default-chain because it violates the first requirement of Definition 7.9. $\mathbf{D}_{235}$ is also no default-chain because it violates the second requirement.

The idea we follow is to find maximal default-chains $\mathbf{D}$ of $\langle W, D\rangle$ that represent extensions. In the example above these are $\mathbf{D}_{123}$, $\mathbf{D}_{213}$, $\mathbf{D}_{124}$ and $\mathbf{D}_{214}$. According to the following theorem a maximal default-chain $\mathbf{D}$ represents an extension if $D \setminus \mathbf{D}$ holds no default that is applicable for $W \cup \mathrm{con}(\mathbf{D})$.

Theorem 7.12 (maximal default-chain as extension)
Let $\mathbf{D}$ be a maximal default-chain of a finite default theory $\langle W, D\rangle$. If there exists no $\delta \in D \setminus \mathbf{D}$ that is applicable for $W \cup \mathrm{con}(\mathbf{D})$ then $\mathrm{Th}(W \cup \mathrm{con}(\mathbf{D}))$ is an extension of $\langle W, D\rangle$.

Proof. Let $\mathbf{D} := (\delta_1, \ldots, \delta_n)$ be a maximal default-chain of $\langle W, D\rangle$ (1) and suppose that there exists no default $\delta \in D \setminus \mathbf{D}$ that is applicable for $W \cup \mathrm{con}(\mathbf{D})$ (2).
Let $E := \mathrm{Th}(W \cup \mathrm{con}(\mathbf{D}))$ and E_i be according to Theorem 5.5, i.e.

$$E_0 := W$$
$$E_{i+1} := \mathrm{Th}(E_i) \cup \{\mathrm{con}(\delta) : \delta \in D, \mathrm{pre}(\delta) \in \mathrm{Th}(E_i) \text{ and } E \cap \neg\mathrm{jus}(\delta) = \emptyset\}.$$

Consider $\delta \in \mathbf{D}$. Then we know from (1) that $E \cap \neg\text{jus}(\delta) = \emptyset$ (3).
Consider $\delta \in D \setminus \mathbf{D}$. Then we know from (2) that we either have $\text{pre}(\delta) \notin E$ (4), or $\text{pre}(\delta) \in E$ and $E \cap \neg\text{jus}(\delta) \neq \emptyset$ (5). If (4) holds then $\text{pre}(\delta) \notin \text{Th}(E_i)$ for an arbitrary i because $\text{Th}(E_i) \subseteq E$. If (5) holds then we trivially have $E \cap \neg\text{jus}(\delta) \neq \emptyset$. Hence the defaults in $D \setminus \mathbf{D}$ can be ignored in the definition of E_{i+1}. Together with (3) this yields

$$E_{i+1} = \text{Th}(E_i) \cup \{\text{con}(\delta) : \delta \in \mathbf{D}, \text{pre}(\delta) \in \text{Th}(E_i)\} \text{ thus } \bigcup_{i=0}^{\infty} E_i \subseteq E.$$

We now show by induction on i that $\text{con}(\delta_i) \in E_i$.

$i = 1$**:** Then we know from (1) that $W \vdash \text{pre}(\delta_1)$, i.e. $\text{pre}(\delta_i) \in \text{Th}(E_0)$ hence $\text{con}(\delta_1) \in E_1$.

$i = j + 1$**:** For $k \leq j$ we know by induction hypothesis that $\text{con}(\delta_k) \in E_k$. Since $E_i \subseteq E_{i+1}$ we have $W \in E_j$ (i) and $\text{con}(\delta_k) \in E_j$ (ii) for $k \leq j$. From (1) we know that $W \cup \text{con}(\delta_1), \ldots, \text{con}(\delta_j) \vdash \text{pre}(\delta_{j+1})$. With (i) and (ii) this yields $\text{pre}(\delta_{j+1}) \in \text{Th}(E_j)$ and thus $\text{con}(\delta_{j+1}) \in E_{j+1}$.

Hence $E \subseteq \bigcup_{i=0}^{\infty} E_i$ and thus $E = \bigcup_{i=0}^{\infty} E_i$, i.e. $E \in \text{Ext}(W, D)$. □

Remark 7.13 (non-maximal default-chains as extensions)
It is possible that a non-maximal default-chain $\mathbf{D}$ of $\langle W, D\rangle$ corresponds to an extension E of $\langle W, D\rangle$, i.e. $E = \text{Th}(W \cup \text{con}(\mathbf{D}))$. This is because generally not every appendable default has a yet unknown consequent.
For example let $W := \{p, q\}$ and $D := \left\{\frac{p : r}{r}, \frac{q : r}{r}\right\}$. Then $\mathbf{D} := \left(\frac{p : r}{r}\right)$ is a non-maximal default-chain of $\langle W, D\rangle$ and $\text{Th}(W \cup \text{con}(\mathbf{D}))$ is the only extension of $\langle W, D\rangle$.
A non-maximal default-chain that corresponds to an extension can easily be extended to a maximal default-chain by appending all remaining applicable defaults to it. The order in which they are appended is arbitrary.

The next theorem ensures that for each extension there exists a corresponding maximal default-chain.

Theorem 7.14 (extension as maximal default-chain)
Let E be an extension of a finite default theory $\langle W, D\rangle$. Then there exists a maximal default-chain $\mathbf{D}$ of $\langle W, D\rangle$ such that $E = \text{Th}(W \cup \text{con}(\mathbf{D}))$.

Proof. Let E be an extension of a finite default theory $\langle W, D\rangle$ and let E_i be according to Theorem 5.5, i.e.

$$E_0 := W$$
$$E_{i+1} := \text{Th}(E_i) \cup \{\text{con}(\delta) : \delta \in D, \text{pre}(\delta) \in \text{Th}(E_i) \text{ and } E \cap \neg\text{jus}(\delta) = \emptyset\}.$$

Let n be the smallest number such that $E_{n+1} = \mathrm{Th}(E_n) = E$. Because $\langle W, D\rangle$ is finite we know that such an n exists. Let $D_1, \ldots, D_n$ be the disjoints non-empty sets of defaults defined as follows.

$$D_i := \{\delta \in D : \mathrm{pre}(\delta) \in \mathrm{Th}(E_i) \text{ and } E \cap \neg\mathrm{jus}(\delta) = \emptyset\} \setminus \bigcup_{j=1}^{i-1} D_j$$

Then D_i contains exactly those defaults whose conclusions define the second part in the definition of E_{i+1}, i.e.

$$\mathrm{Th}(E_{i+1}) = \mathrm{Th}(W \cup \bigcup_{j=1}^{i} \mathrm{con}(D_j)), \text{ hence } E = \mathrm{Th}(W \cup \bigcup_{i=1}^{n} \mathrm{con}(D_i)).$$

For $\delta \in D_i$ we have $\mathrm{jus}(\delta) \cap E = \emptyset$ and $W, \mathrm{con}(D_1), \ldots, \mathrm{con}(D_{i-1}) \vdash \mathrm{pre}(\delta)$. Let $D_i = \{\delta_{i,1}, \ldots, \delta_{i,m_i}\}$ for $1 \leq i \leq n$. Then the following sequence defines a default-chain for $\langle W, D\rangle$.

$$\mathbf{D} := \langle \delta_{1,1}, \ldots, \delta_{1,m_1}, \delta_{2,1}, \ldots, \delta_{2,m_2}, \ldots, \delta_{n,1}, \ldots, \delta_{n,m_n} \rangle$$

$\mathbf{D}$ contains all defaults of D that are applicable for E. Because E is an extension of $\langle W, D\rangle$ there exists no other applicable default for E besides those in $\mathbf{D}$. Hence $\mathbf{D}$ is maximal. □

A maximal default-chain that corresponds to an extension must not have the form as given in the proof above. This is because a default-chain is defined element-wise while E_{i+1} is defined set-wise.

Let $\langle W, D\rangle := \left\langle \emptyset, \left\{ \frac{:p}{p}, \frac{:q}{q}, \frac{p:r}{r} \right\} \right\rangle$. Then $\left(\frac{:p}{p}, \frac{p:r}{r}, \frac{:q}{q} \right)$ is a maximal default-chain that corresponds to the only extension $E = \mathrm{Th}(\mathrm{con}(D))$ of $\langle W, D\rangle$ but does not have the form as given in the proof above.

7.3.1 Computing Extensions

From Theorem 7.14 we know that for every extension of $\langle W, D\rangle$ there exists a corresponding maximal default-chain of $\langle W, D\rangle$. Furthermore Theorem 7.12 gives us the requirements a maximal default-chain must have to correspond to an extension. A simple method to compute the extensions of a default theory is therefore to compute first all its maximal default-chains and then compute from them the extensions according to Theorem 7.12. Since a default-chain is built up element-wise, an algorithm to compute all maximal default-chains can easily be given (see Algorithm 24).

In our example we have seen that two different default-chains can be set-equal. It is easy to see that if two set-equal default-chains $\mathbf{D}_1$ and $\mathbf{D}_2$

Algorithm 24 Computing maximal default-chains

```
1: function maxDefChains(⟨W, D⟩)
2:     return extendChain(() , W, D)

3: function extendChain((D, W, D))
4:     D⁺ := {δ ∈ D : δ appendable to D for W}
5:     if D⁺ = ∅ then
6:         return {D}
7:     else
8:         result := {}
9:         for δ ∈ D⁺ do
10:            result := result ∪ extendChain(D ∘ δ, W, D \ {δ})
11:        return result
```

correspond to the extensions E_1 and E_2 then these extensions must be equal. To compute the extension of a default theory from the maximal default-chains it is thus sufficient to consider only one representative among the set-equal maximal default-chains. A way to find just one representative is based on the following observation.

Consider a default-chain $\mathbf{D}$ of $\langle W, D\rangle$ and suppose that a maximal default-chain $\mathbf{D}_1$ includes the elements of $\mathbf{D}$. Then it is easy to see that there exists a maximal default-chain $\mathbf{D}_2$ that is set-equal to $\mathbf{D}_1$ and has $\mathbf{D}$ as a prefix. To find a representative of all set-equal maximal default-chains that include $\mathbf{D}$ it is thus sufficient to consider only those that start with $\mathbf{D}$.

Now let $\delta \in D \backslash \mathbf{D}$ be appendable to $\mathbf{D}$ for W. According to the remark above we have to consider two options to extend $\mathbf{D}$ to a maximal default-chain.

1. We calculate the maximal default-chains that start with $\mathbf{D} \circ \delta$, i.e. we recursively proceed the search with $\mathbf{D} \circ \delta$.

2. We calculate the maximal default-chains that start with $\mathbf{D}$ but do not contain δ, i.e. we mark δ to be not appendable for following recursive calls and recursively proceed our search with $\mathbf{D}$.

By marking defaults to be not appendable the recursion may stop before a maximal default-chain is reached. On recursion end we thus first check whether the marked defaults turned out to be not appendable anymore in the meantime. If this is the case then the default-chain is maximal and we can decide according to Theorem 7.12 whether it represents an extension. Otherwise we know that a maximal default-chain that includes the elements of the current default-chain is considered in another recursion branch of our algorithm.

An algorithm that follows this strategy to compute the extensions of a default theory is given in Algorithm 25.

Algorithm 25 Computing extensions of a default theory

```
 1: function EXTENSIONS(W, D, D, D_out)
 2:     find δ ∈ D that is applicable for W ∪ con(D)
 3:     if such a δ exists then
 4:         D' := D \ {δ}
 5:         if δ is appendable to D for W then
 6:             return EXTENSIONS(W, D', D ∘ δ, D_out) ∪
 7:                    EXTENSIONS(W, D', D, D_out ∪ {δ})
 8:         else
 9:             return EXTENSIONS(W, D', D, D_out ∪ {δ})
10:     else
11:         if no δ ∈ D_out is applicable for W ∪ D then
12:             return {{W ∪ pre(D)}}
13:         else
14:             return {}
```

The function EXTENSIONS takes four arguments. The base theory W, the set of defaults D that may be appendable, the default-chain $\mathbf{D}$ that is to be maximized, and the set of defaults D_{out} that were marked to be not appendable or that were applicable but not appendable at a lower recursion level.

The algorithm distinguishes between applicable and appendable defaults to avoid redundancies.
A default that is applicable for $W \cup \mathbf{D}$ but not appendable to $\mathbf{D}$ for W is known to be not appendable in following recursive calls. It therefore does not have to be considered there anymore. In order that an extended version $\mathbf{D}'$ of the current default-chain $\mathbf{D}$ represents an extension, such a default must turn out to be not applicable for $\mathbf{D}'$.
An appendable default that is marked to be not appendable must also turn out to be not applicable for an extended version $\mathbf{D}'$ of the current default-chain $\mathbf{D}$ in order that $\mathbf{D}'$ is maximal and represents an extension.
We therefore treat both kinds of defaults equally and add them to the set of defaults D_{out} that have to be not applicable on recursion end to confirm a default-chain to be maximal and represent an extension.

There is a further optimization we use that is not visible in the pseudocode. When searching for an applicable default on line 2 we may encounter defaults whose prerequisites follows from $W \cup \text{con}(\mathbf{D})$ but whose justifications are not met. It is clear that such defaults need not be checked anymore in following recursive calls.

Example

A possible run of EXTENSIONS$(W, D, \mathbf{D}, D_{\text{out}})$ for example 7.11 is given in Figure 7.3. There the first appendable default in D is marked green. On recursion end a maximal default-chain is marked green and a non-maximal one red. If a maximal default-chain represents an extension then its recursion label is marked green otherwise it is marked red.
There are exactly three maximal default-chains. None of them is set-equal to an other. The first two (1.1.1 and 1.1.2.1) represent extensions while the last of them (2.2.1) does not because there are defaults in D_{out} that are applicable but not appendable.

7.4 Experimental Results

In this section we compare the different proving approaches on some scalable problems. The problems mainly aim to point out the advantages and disadvantages of the approaches. All problems were computed under Debian Linux 4.0 on an AMD Sempron 2600+ with 1.5 GB RAM.

To prove whether a default sequent is credulously valid we have investigated three main approaches.

1. Do backward application of the deduction rules and prefer rule (cD4) to (cD5). For this approach we have investigated the following provers. PR4, implementing no improvements (cf. Algorithm 17, p. 221). PR4m, implementing the approach with minimal requirements (cf. Algorithm 18, p. 223). PR4r, implementing residual improvement (cf. Algorithm 20, p. 227). PR4mr, implementing the approach with minimal requirements and residual improvement.

2. Do backward application of the deduction rules and prefer rule (cD5) to (cD4). For this approach we have investigated the following provers. PR5, implementing no improvements. PR5m, implementing the approach with minimal requirements. PR5r, implementing residual improvement and PR5mr, implementing the approach with minimal requirements and residual improvement.

3. Iterate through the extensions according to Algorithm 25 and check with the CPC prover whether the succedent is in the extension. The prover that follows this strategy is called PRext.

To prove whether a default sequent is skeptically valid we have also investigated three main approaches.

1. Do backward application of the deduction rules by partitioning the

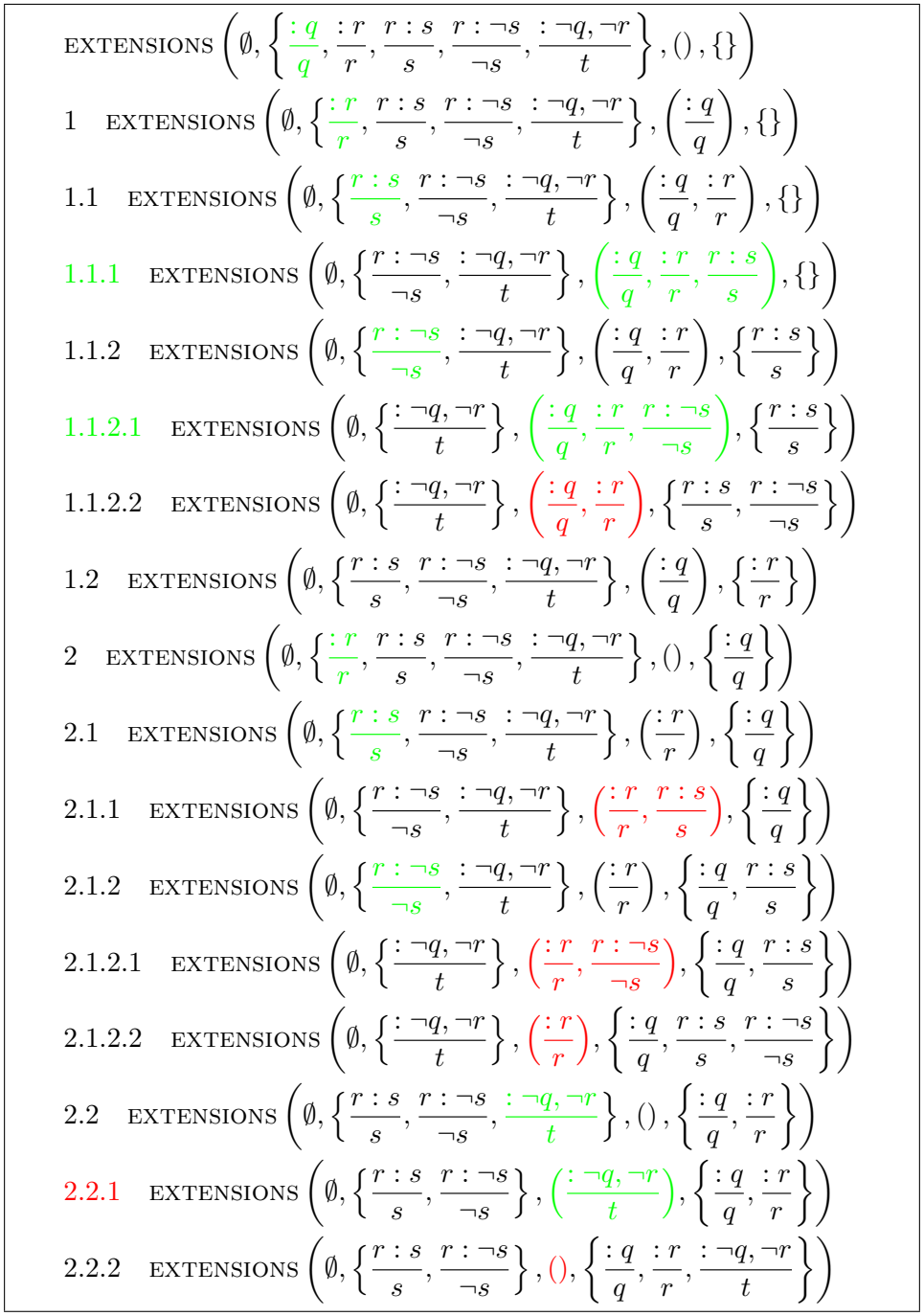


Figure 7.3: Example run of EXTENSIONS$(W, D, \mathbf{D}, D_{\text{out}})$

defaults and prefer dropping a default, i.e. we start with a maximal D_{out}. For this approach we have investigated the following provers. PR0, implementing no improvements. PR0r, implementing residual improvement. PR0u, implementing use-check. PR0ru, implementing residual improvement and use-check.

2. Do backward application of the deduction rules by partitioning the defaults and prefer keeping a default, i.e. we start with a maximal D_{in}. For this approach we have investigated the following provers. PR1, implementing no improvements, PR1r, implementing residual improvement, PR1u, implementing use-check and PR1ru, implementing residual improvement and use-check.

3. Iterate through the extensions according to Algorithm 25 and check with the CPC prover whether the succedent is in the extension. The prover that follows this strategy is called PRext.

For each problem we investigate a sorted and a random scenario, that is we use a sorted and a random order in which the defaults are processed. To have comparable results for the random scenario we initialize the random number generator with the number of residues in the theory before shuffling.

Many of the provers are slow for most problems. We therefore examine all provers only for the first problem and discuss for later problems only the fast provers.

We use graphics to visualize the given default theories. To depict the default $A : B_1, B_2/C$ we use labeled arrows, where the label corresponds to the justification of the default: $A \xrightarrow{B_1,B_2} C$. To depict that the formula A is in the base theory we draw a circle around it: $\textcircled{A}$.

7.4.1 Problem 1: Chain of Defaults

We start with a very simple residue theory that is defined as follows.

$$T_1(n) := \{p_1\} \cup \left\{ \frac{p_i : p_{i+1}, q}{p_{i+1}} : 1 \leq i \leq n \right\}$$

Because we have special cases for normal theories, the justification q is added to operate over a non-normal default theory.

In this theory we have a chain of defaults and thus the single extension $\text{Th}(p_1, \ldots, p_{n+1})$. Depicted as a graph $T_1(n)$ looks as follows.

$$\textcircled{p_1} \xrightarrow{p_2,q} p_2 \xrightarrow{p_3,q} \cdots \xrightarrow{p_{n+1},q} p_{n+1}$$

In the sorted scenario we use the index of the prerequisite as sort key.

Proving $T_1(n) \supset p_{\frac{n}{2}+1}$

The problem is set in a way that there exists one minimal requirement that consists of the first $\frac{n}{2}$ defaults.

Credulous Provers
Since all defaults of $T_1(n)$ are generating for the only extension of $T_1(n)$, the provers that prefer (cD5) follow for this theory an optimal strategy to find an extension that contains $p_{\frac{n}{2}+1}$. The results are accordingly (cf. figures 7.4 and 7.5). We analyze the slow provers for $n = 20$ and the fast prover for $n = 500$ (cf. tables 7.2 and 7.3).

PR4 and PR4g perform worst because they have to do maximal backtracking. PR4 is a bit faster than PR4g. This indicates that residual improvement is for this example counterproductive. For $n = 20$ we can omit about a thousand PRC proofs but this does not weight up the overhead introduced by residual improvement.
PR4m and PR4mg perform somewhat better than PR4 and PR4g because they don't apply (cD4) on defaults that are in the minimal requirement. The figures for $n = 20$ confirm this. In the sorted scenario (cD4) can be omitted 10 times. This results in having far less (cD4) and (cD5) rule applications and thus fewer unsuccessful PRC proofs. The number of successful PRRC proofs — they represent the verification of the justifications — is the same for all four provers. Residual improvement is of no use for PR4mg. Therefore the prover only introduces additional overhead for this problem.
In the random scenario the four slow provers show the same performance. Because the defaults of the minimal requirement are not processed first anymore, the figures of omitted (cD4) rules and with them those of unsuccessful (cD5) rule applications change for PR4m and PR4mg. Compared to the sorted scenario we now have 1283 instead of 10 omitted (cD4) rules and 2286 instead of 1013 unsuccessful (cD5) rule applications.

PRext performs quite good, processing the defaults in sorted order is in fact optimal for this problem. Nevertheless it can not compete with the backward rule based provers that prefer (cD5). For them the sorted order is also optimal. PR5 and PR5m perform best while PR5r and PR5mr perform marginally worse due to the overhead produced by residual improvement.
The figures for the sorted scenario confirm the good performances of all provers. There are no failed CPC and CPRC proofs for PRext. Hence checking whether a default is applicable and addable never fails. PR5, PR5m, PR5r and PR5mr also have optimal figures.
In the random scenario the figures only change for PRext. There we have for $n = 500$ about 60 000 failed CPC proofs. This is because the defaults are no longer in the order in which they need to be added to the default-chain.

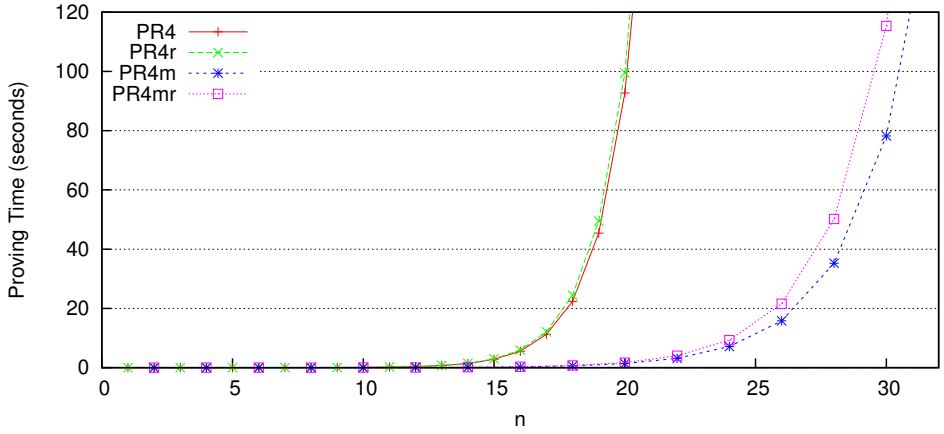


Figure 7.4: Proving time of ; $T_1(n) \supset p_{\frac{n}{2}+1}$ (sorted order).

	PRC		PRRC		(cD4)			(cD5)	
	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	–	$\top$	$\bot$
PR4	1E3	1E6	31E3	0	0	1E6	0	20	1E6
PR4r	1	1E6	31E3	0	0	1E6	0	20	1E6
PR4m, PR4mr	0	2E3	31E3	0	0	1E3	10	20	1E3

Table 7.2: Figures of proving $T_1(15) \supset p_8$ (sorted order)

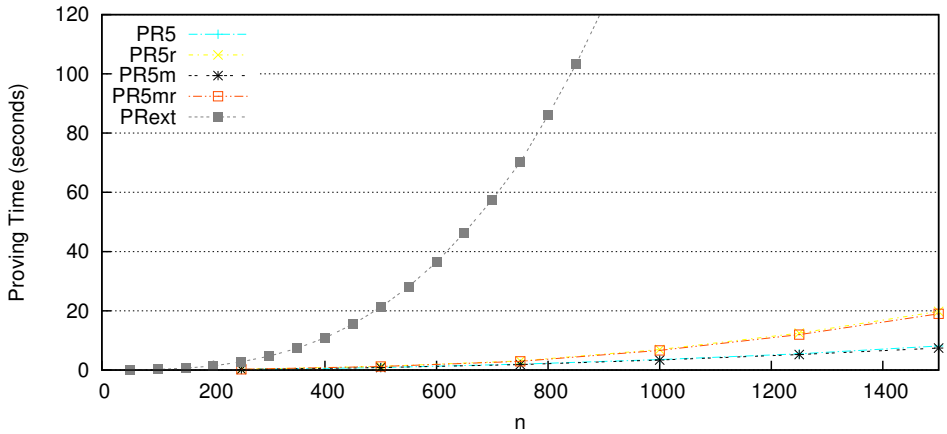


Figure 7.5: Proving time of $T_1(n) \supset p_{\frac{n}{2}+1}$ (sorted order).

	PRC		PRRC		(cD5)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$
PR5, PR5r	1	0	1000	0	500	0				
PR5m, PR5mr	0	0	1000	0	500	0				
PR5ext							501	0	251 500	0

Table 7.3: Figures of proving $T_1(500) \supset p_{251}$ (sorted order)

Skeptical Provers
For skeptical default entailment there is no optimal strategy for this problem since being fast in finding one extension that contains $p_{\frac{n}{2}+1}$ does not imply that $p_{\frac{n}{2}+1}$ is in all extensions. The question is whether the optimizations turn out to work well for the corresponding search strategy.

The strategy to prefer dropping a default turns out to be inefficient. All provers that follow this strategy and PR1 show the same poor performance (cf. Figure 7.6). For $n = 15$ residual improvement allows us to omit only 255 CPC proofs and use-check is only successful in 8 of 32 520 cases (cf. Table 7.4).
PR1r is somewhat faster than the provers mentioned above. This is because due to residual improvement most of the successful PRC and unsuccessful PRRC proofs can be omitted. However, since residual improvement does not allow us to omit whole branches in the proof search tree, the impact of this optimization is small.
There is no significant changes in the random scenario.

PR1ru and PR1ru perform good (cf. Figure 7.7). The strategy to prefer keeping a default together with use-check is quite successful. According to the figures for $n = 300$ the prover only branches in 150 of 33 975 cases (cf. Table 7.5).
The extension based prover PRext performs best. Compared to the credulous case it does not drop very much in performance. This is because when we deliberately mark an appendable default δ to be not appendable then no other default turns out to be applicable and hence the effort to find an default-chain that does not contain δ is rather small.
In the random scenario the fast provers perform equally well. The figures for PRext are exactly the same while for PR1u and PR1ru the number of (sD4) rule applications drops from 32 520 to 22 507 and the number of branches remains at 150. This indicates that the sorted order is not optimal for this problem. Further investigations have shown that when proving p_2 instead of $p_{\frac{n}{2}+1}$ then PR1u and PR1ru perform much better and prove the problem in the sorted scenario for $n = 800$ in about 2 seconds while the performance of PRext remains the same.

Proving $T_1(n) \supset p_2 \vee p_3 \vee \cdots \vee p_n \vee p_{n+1}$

The next problem we investigate is to prove the disjunction of the defaults conclusions.

Credulous Provers
The credulous provers show little performance difference to the previous problem. PR4, PR4g, PR4m and PR4mg perform equally bad and reach the

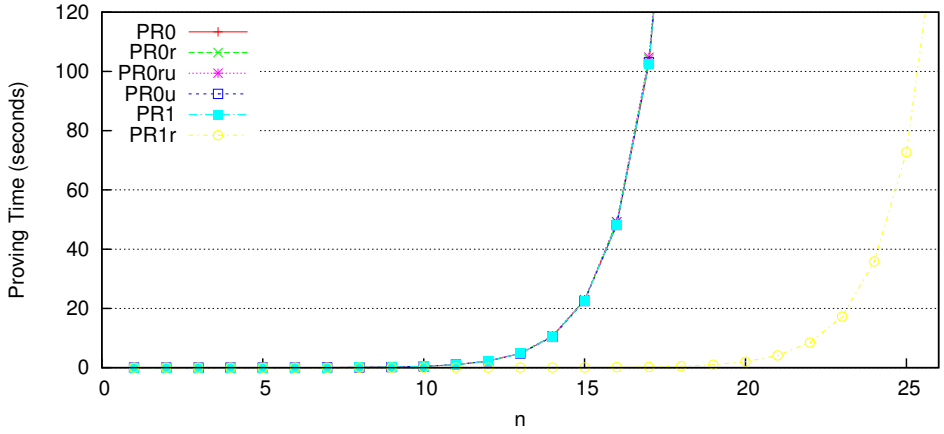


Figure 7.6: Proving time of $;T_1(n) \supset p_{\frac{n}{2}+1}$ (sorted order).

	PRC		PRRC		(sD4)	
	$\top$	$\bot$	$\top$	$\bot$	in	out
PR0, PR1	256	518 400	65 024	0	32 767	32 767
PR0r	1	518 400	65 024	0	32 767	32 767
PR0ru, PR0u	1	518 400	65 024	0	32 512	32 520
PR1r	1	203	14	0	32 767	32 767

Table 7.4: Figures of proving $T_1(15) \supset p_8$ (sorted order)

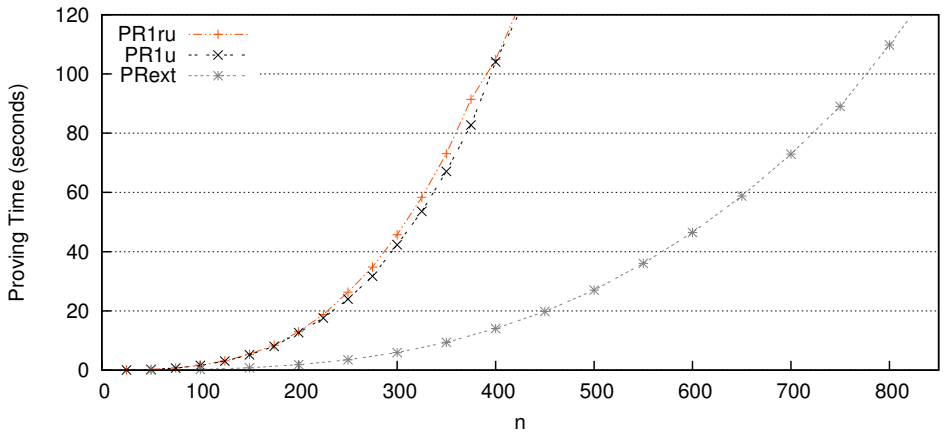


Figure 7.7: Proving time of $;T_1(n) \supset p_{\frac{n}{2}+1}$ (sorted order).

	PRC		PRRC		(sD4)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	in	out	$\top$	$\bot$	$\top$	$\bot$
PR1ru, PR1u	1	90E3	300	0	34E3	150				
PR1ext							301	45E3	92E3	0

Table 7.5: Figures of proving $T_1(300) \supset p_{151}$ (sorted order)

two minute limit for $n = 17$ or $n = 18$. The performance of PR5, PR5g, PR5m, PR5mg and PR5ext is the same as for the previous problem.

Skeptical Provers
The slow skeptical provers show almost the same performance as for the previous problem. We therefore only discuss the performance of the fast skeptical provers in depth.

PRext shows for the sorted and random scenario the same performance as for the previous problem. This is not astonishing since verifying whether the succedent is in an extension is rather easy and the time to iterate through all extensions only depends on the default theory and the processing order of the defaults. And these two factors are the same as for the previous problem.

PR1u and PR1ru however show very different performances for the sorted and random scenario (cf. figures 7.8 and 7.9). In the sorted scenario the performance is comparable to the performance of the previous problem. In the random scenario the performance increases but is rather unsteady for PR1u and somewhat unsteady for PR1ru.

The different behavior of the sorted and the random scenario indicates that the sorted scenario reflects a bad case for the provers with use-check. A closer look reveals two facts.

1. *The formula in the succedent reflects the worst case for use-check.*
   If an application of (sD1) succeeds then use-check depends heavily on the axioms encountered in the proof of the premise. Because $\vee$ associates to the left the prover detects the atoms in $p_2 \vee \cdots \vee p_{n+1}$ from right to left. Hence p_{n+1} has the highest and p_2 the lowest preference to form an axiom.
   Suppose that $p_m : p_{m+1}, q/p_{m+1}$ is the first skipped default. Because of the above preference to form an atom the first $m-1$ defaults will be marked as used, while it would be sufficient to only mark $p_1 : p_2, q/p_2$ as used. The prover thus always uses the maximal possible dependencies for the given succedent.

2. *The sorted order reflects a worst case.*
   An optimal processing order of the defaults for this problem is to treat the second default last. If we do so then the first partition will have all defaults in D_{in} and will succeed with (sD1) marking all defaults as used. The second partition then has all but the second default in D_{in} and succeeds with (sD1) marking only the first default as used. Because (sD3) is not used for the second partition use-check then drops the dependency information of the first partition. In the following all but one branching can be omitted.
   The branching that has to be processed is the one that treats the first

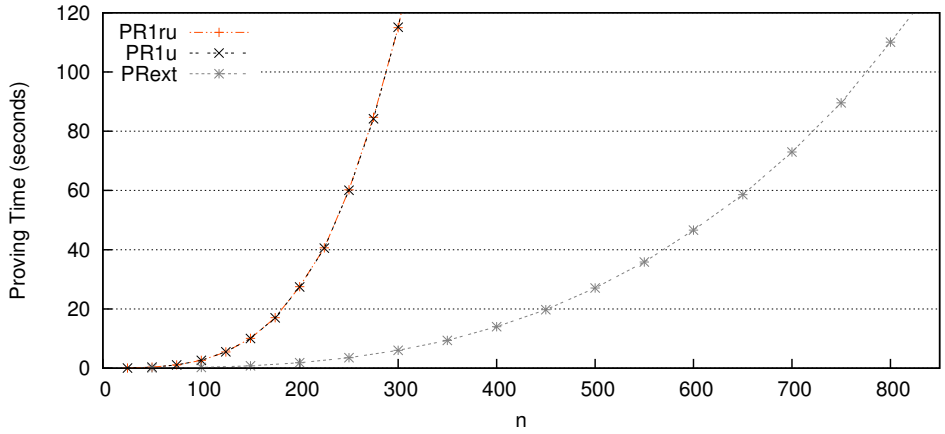


Figure 7.8: Proving time of $; T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (sorted order).

	PRC		PRRC		(sD4)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	in	out	$\top$	$\bot$	$\top$	$\bot$
PR1ru, PR1u	300	599	2	0	45E3	300				
PR1ext							301	45E3	92E3	0

Table 7.6: Figures of proving $T_1(300) \supset p_2 \vee \cdots \vee p_{301}$ (sorted order)

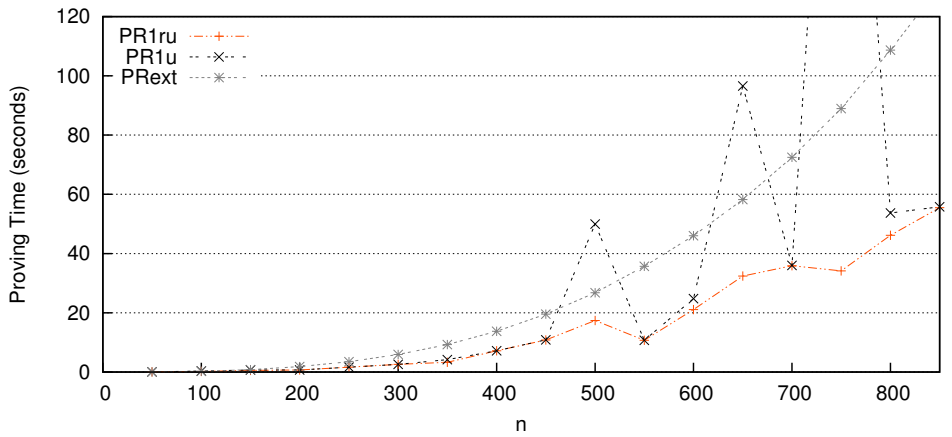


Figure 7.9: Proving time of $; T_1(n) \supset p_2 \vee \cdots \vee p_{n+1}$ (random order).

	PRC		PRRC		(sD4)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	in	out	$\top$	$\bot$	$\top$	$\bot$
PR1ru	10	999	2	0	1145	11				
PR1u	18	9960	20	0	4918	27				
PR1ext							501	125E3	253E3	0

Table 7.7: Figures of proving $T_1(500) \supset p_2 \vee \cdots \vee p_{501}$ (random order)

default. There the first default is skipped. The first partitioning that is then encountered has thus all but the first default in D_{in}. This partitioning succeeds with (sD3) and according to use-check no more branching will hence be necessary in the following.

The explanation of the optimal processing implies the choice of a good processing order. If a default with a low sort key is processed late by the prover then little dependencies are introduced. This effects that the impact of use-check is good. Processing the defaults in a random order leads to such situation. This explains the good performance in the random scenario.

A further interesting observation for this problem is that in the random scenario the combination of use-check and residual improvement leads to a steadier behavior. This is because we predict the success of a (sD1) application if we know that (sD1) succeeded for a partition whose set D'_{in} of selected defaults is a subset of the currently selected defaults D_{in}. For use-check we then take the use-check information of the know result. And because $D'_{\text{in}} \subset D_{\text{in}}$ we then do not necessarily introduce the maximal possible dependencies.

Proving $T_1(n) \supset p_{n+1} \vee p_n \vee \cdots \vee p_3 \vee p_2$

If we revert the order in the disjunction of the succedent then PR1r and PR1ru show their best case behavior and are thus clearly better than PRext (cf. Figure 7.10). The reason for this is clear. Now p_2 has the highest preference to form the axiom. The prover thus introduces only minimal dependencies. As a consequence the performance is independent of the processing order of the defaults. The random scenario thus leads to the same performance.

Proving $T_1(n) \supset p_2 \wedge p_3 \wedge \cdots \wedge p_n \wedge p_{n+1}$

If we prove the conjunction $p_2 \wedge \cdots \wedge p_{n+1}$ of the default's consequents then the performance of PR1u and PR1ru is comparable to their performance when proving the corresponding disjunction $p_2 \vee \cdots p_{n+1}$ in the sorted scenario (cf. Figure 7.11).

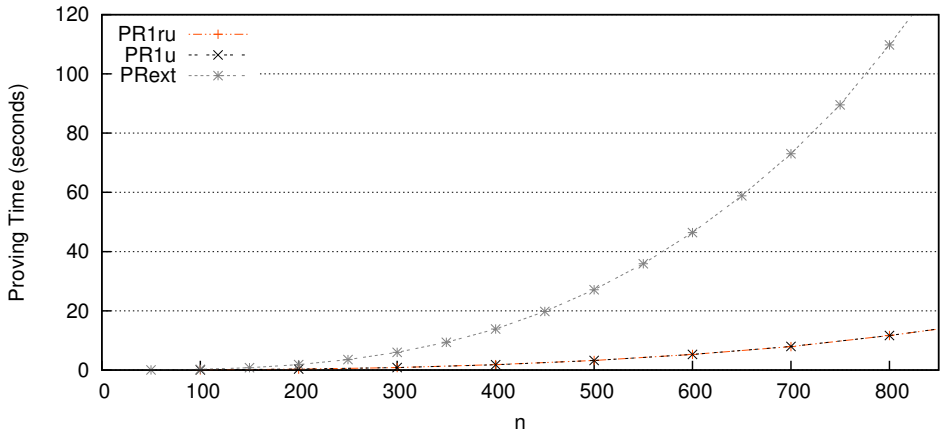


Figure 7.10: Proving time of ; $T_1(n) \supset p_{n+1} \vee \cdots \vee p_2$ (sorted order).

	PRC		PRRC		(sD4)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	in	out	$\top$	$\bot$	$\top$	$\bot$
PR1ru, PR1u	1	999	2	0	999	1				
PR1ext							501	125E3	253E3	0

Table 7.8: Figures of proving $T_1(500) \supset p_{501} \vee \cdots \vee p_2$ (sorted order)

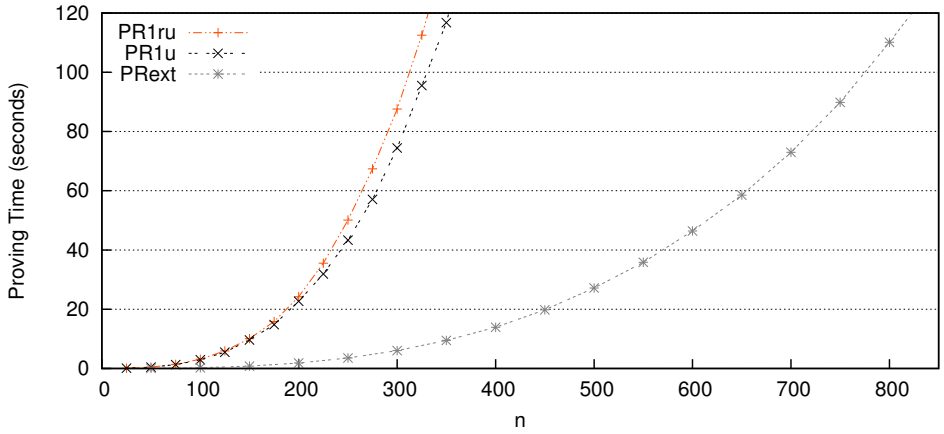


Figure 7.11: Proving time of ; $T_1(n) \supset p_2 \wedge \cdots \wedge p_{n+1}$ (sorted order).

	PRC		PRRC		(sD4)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	in	out	$\top$	$\bot$	$\top$	$\bot$
PR1ru, PR1u	1	180E3	600	0	45E3	300				
PR1ext							301	45E3	92E3	0

Table 7.9: Figures of proving $T_1(300) \supset p_2 \wedge \cdots \wedge p_{301}$ (sorted order)

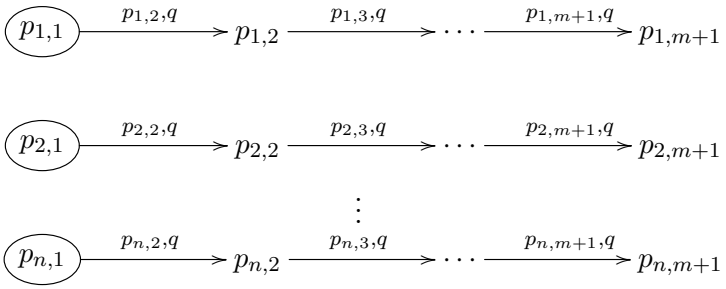
7.4.2 Problem 2: Short Chains of Defaults

We continue with a simple default theory that is defined as follows:

$$T_2(n,m) := \{p_{i,1} : 1 \leq i \leq n\} \cup \bigcup_{i=1}^{n} \left\{ \frac{p_{i,j} : p_{i,j+1}, q}{p_{i,j+1}} : 1 \leq j \leq m \right\}$$

Again we add q to the justifications in order to force the non-normal proving techniques.

$T_2(n)$ has n chains of m defaults and the prerequisites of the head of each chain are in the base theory of $T_2(n)$. The theory therefore has exactly one extension that contains all consequents of its defaults. Depicted as a graph $T_2(n,m)$ looks as follows.



In the sorted scenario we use the prerequisite as primary and the consequent as secondary sort key. The sorting of the propositions is done lexicographically according to their indexes.

Proving $T_2(n) \supset p_{1,6} \vee p_{2,6} \vee \cdots \vee p_{n-1,6} \vee p_{n,6}$

The first problem we investigate for $T_2(n)$ is to prove the disjunction of the consequents of the last elements in the chains.

Credulous Provers
The theory $T_2(n)$ is very similar to $T_1(n)$. Since all defaults are generating we again have the case that the theory is optimal for provers that prefer keeping a default, i.e. adding it to D_{in}. The picture for this problem is therefore similar for the first problem of $T_1(n)$. We thus omit discussing this and the next problem for the credulous provers.

Skeptical Provers
For this problem all provers show a very poor performance (cf. Figure 7.12).

PRext performs so bad because deliberately marking an appendable default to be not appendable causes only at most 4 other defaults to be no longer applicable. The search tree is thus not reduced substantially.

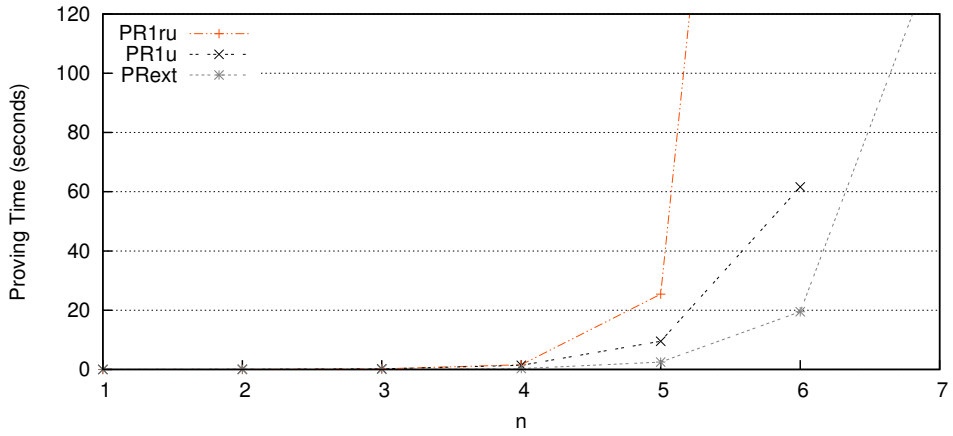


Figure 7.12: Proving time of ; $T_2(n) \supset p_{1,6} \vee \cdots \vee p_{n,6}$ (sorted order).

	PRC		PRRC		(sD4)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	in	out	$\top$	$\bot$	$\top$	$\bot$
PR1ru	5	128E3	6250	0	13E3	4155				
PR1u	1031	128E3	6250	0	13E3	4155				
PR1ext							7776	65E3	226E3	0

Table 7.10: Figures of proving $T_2(5) \supset p_{1,6} \vee \cdots \vee p_{5,6}$ (sorted order)

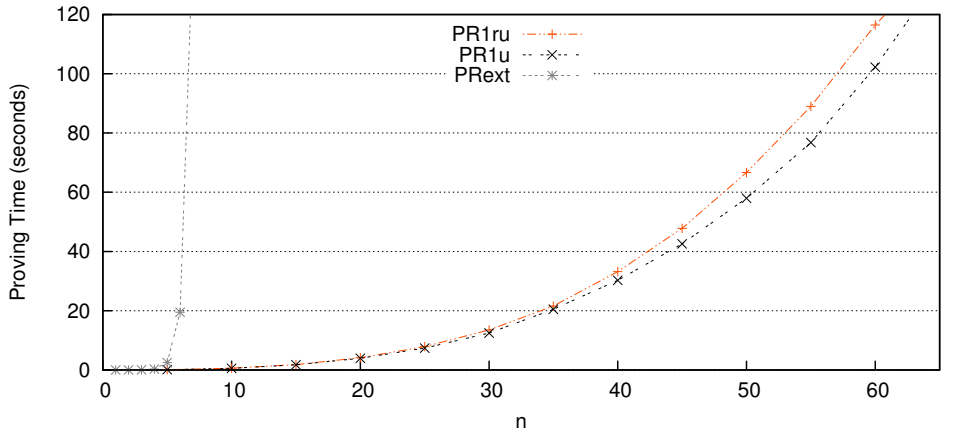


Figure 7.13: Proving time of ; $T_2(n) \supset p_{1,6} \wedge \cdots \wedge p_{n,6}$ (sorted order).

	PRC		PRRC		(sD4)	
	$\top$	$\bot$	$\top$	$\bot$	in	out
PR1ru, PR1u	1	124 750	500	0	31 375	250

Table 7.11: Figures of proving $T_2(50) \supset p_{1,6} \wedge \cdots \wedge p_{50,6}$ (sorted order)

The performance of PR1u shows that also use-check is of no use to speed up the proving process. PR1ru performs even worse than PR1u which implies that residual improvement introduces more overhead than benefit to the proving process. The reason that use-check is not so successful it mainly because only five defaults are really needed to prove the sequent but there are m possibilities to choose them. The ratio of successful to unsuccessful use-checks is thus not so high as in previous examples where use-check lead to good results (cf. Table 7.10). Hence the effect of use-check is for this problem rather small.

The performance of the provers remains the same in the random scenario or if we invert the order in the disjunction of the succedent. The processing order of the defaults is thus not relevant for the performance of the provers.

Proving $T_2(n) \supset p_{1,6} \wedge p_{2,6} \wedge \cdots \wedge p_{n-1,6} \wedge p_{n,6}$

The next problem we investigate for $T_2(n)$ is to prove the conjunction of the consequents of the last elements in the chains.

As mentioned already in the previous problem, we omit discussing the credulous provers here because we gain no further insight to the provers from this problem.

Skeptical Provers
Compared to the previous problem PRext shows the same poor performance while PR1u and PR1ru perform much better (cf. Figure 7.13)

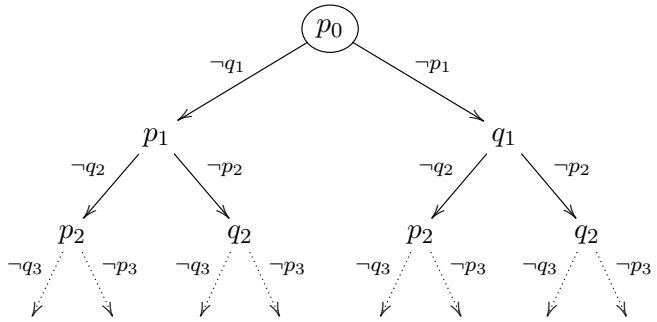
The observation that PRext shows the same performance is not astonishing since the main work of this provers for this problem is to iterate through the extensions of the theory and this is independent of the formula that is to be proven.
The provers with use-check perform so good because skipping a default leads to a selection for which (sD1) and (sD2) fail and (sD3) succeeds. Skipping further defaults is thus known to also succeed according to use-check and hence we branch for each default just once. The figures for $n = 50$ confirm this observation (cf. Table 7.11).

7.4.3 Exponentially Many Extensions

The next theory we are going to inspect has exponentially many extensions and is defined as follows.

$$T_3(n,m) := \{p_0\} \cup \left\{ \frac{p_0 : \neg q_1}{p_1}, \frac{p_0 : \neg p_1}{q_1} \right\} \cup \left\{ \frac{p_{i-1} : \neg q_i}{p_i} : 1 < i < n \right\} \cup \left\{ \frac{p_{i-1} : \neg p_i}{q_i} : 1 < i < n \right\} \cup \left\{ \frac{q_{i-1} : \neg q_i}{p_i} : 1 < i < n \right\} \cup \left\{ \frac{q_{i-1} : \neg p_i}{q_i} : 1 < i < n \right\}$$

$T_3(n)$ forms a binary tree that has p_0 in its root and each other node carries p_i or q_i. A node that carries p_i or q_i has p_{i+1} as left and and q_{i+1} as right son. Each connection is labeled with the negation of the other son. Depicted as a graph $T_3(n)$ looks as follows.



From that graph it is easy to see that the theory has exponentially many extensions since every path from the root of the tree to a leaf node represents an extension.

In the sorted scenario we use the prerequisite as primary and the consequent as secondary sort key. The sorting of the propositions is done lexicographically according to their indexes.

Proving $T_3(n) \supset q_1$

The first problem we investigate is to prove q_1. This problem is credulously but not skeptically entailed by $T_3(n)$.

Credulous Provers
Regarding the graphical representing of $T_3(n)$ the prover has to find a path from the root of the tree to a leave in the right subtree of the root node. However, all provers perform rather bad (cf. Figure 7.14).

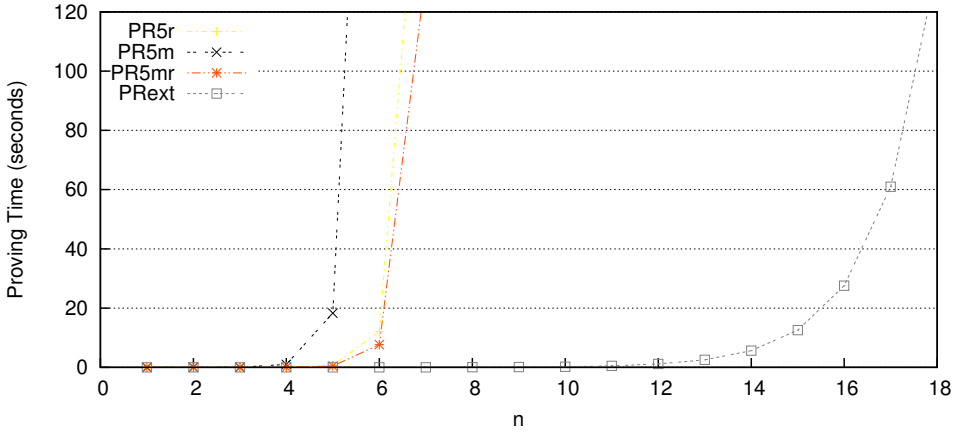


Figure 7.14: Proving time of $T_3(n) \supset q_1$ (sorted order).

	PRC		PRRC		(cD4)			(cD5)	
	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	–	$\top$	$\bot$
PR5m	4105	8193	150E3	83E30	9	87E3	1	9	87E3
PR5mr	4	1	24	90	9	87E3	1	9	87E3
PR5r	5	2	24	90	9	87E3	1	9	87E3

	CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$
PR1ext	400	40E3	21E3	200

Table 7.12: Figures of proving $T_3(5) \supset q_1$ (sorted order)

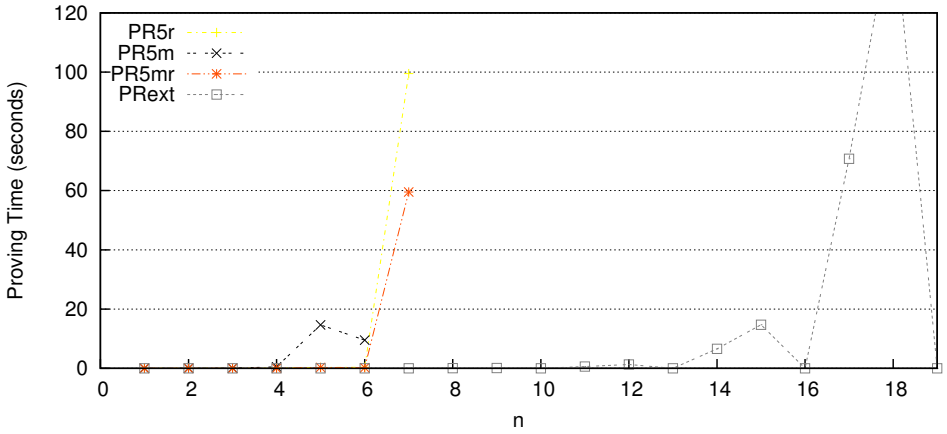


Figure 7.15: Proving time of $T_3(n) \supset q_1$ (random order).

For the backward rule based provers the reason for the bad performance is that each extension is represented by a partition that halves the set of defaults. Hence neither the strategy to first drop nor the one to first keep a default leads to a solution at first. Furthermore the single minimal requirement that contains exactly one default is of little help since it is good to omit backtracking only once (cf. Table 7.12). The impact of residual improvement is also very poor.

For the extension based prover the reason for the bad performance is to find in the given processing order which results in traversing the left subtree of the root node first. In that subtree the paths represent all extensions that contain p_1 but not q_1. Hence the prover iterates through half of the extensions before finding one that contains the formula that is to prover.

In the random scenario the performance of the backward rule based provers remains the same while the performance of PRext is highly unsteady. If $p_0 : \neg p_1/q_1$ is processed before $p_0 : \neg q_1/p_1$ then PRext is very fast, since the first extension it encounters witnesses the formula that is to prove. Otherwise it shows the same poor performance as in the sorted scenario (cf. Figure 7.15).

Skeptical Provers
In the sorted scenario PR1ext performs very good and PR1ru and PR1u perform good (cf. Figure 7.16).

The reason for the good performance of PR1ext is to find in the processing order. Since $p_0 : \neg q_1/p_1$ is processed first, the first extension PR1ext encounters does not contain q_1, it can hence return immediately with failure.
The good performance of the other two provers is to find in the combination of the processing order of the defaults and use-check. According to the figure (cf. Table 7.13) use-check is successful in all but 399 of 159 600 cases.

In the random scenario the performance of all provers drops heavily and they all show an unsteady behavior (cf. Figure 7.17).

The poor and unsteady performance of PRext has the same reason as in the credulous case. If the prover processes $p_0 : \neg q_1/p_1$ before $p_0 : \neg p_1/q_1$ then the first extension encountered by PRext leads to immediate failure, otherwise it first iterates over all extensions that contain q_1 before starting to iterate over the extensions that do not contain it and has thus exponential run time behavior.

For PR1ru and PR1u the processing order is also crucial. An interesting observation is that the combination of use-check with residual improvement leads to somewhat better results.

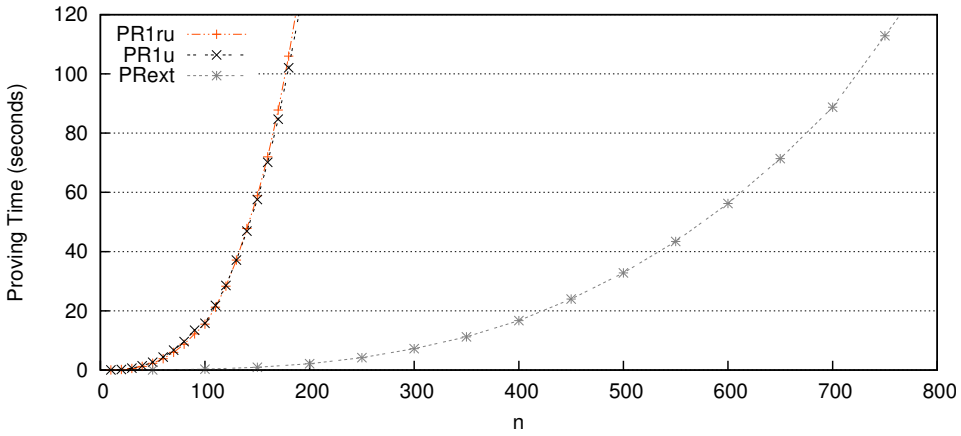


Figure 7.16: Proving time of $T_3(n) \supset q_1$ (sorted order).

	PRC		PRRC		(sD4)		CPC		CPRC	
	$\top$	$\bot$	$\top$	$\bot$	in	out	$\top$	$\bot$	$\top$	$\bot$
PR1ru	399	81E3	1	200	160E3	399				
PR1u	399	81E3	1	399	160E3	399				
PR1ext							400	40E3	21E3	200

Table 7.13: Figures of proving $T_3(200) \supset q_1$ (sorted order)

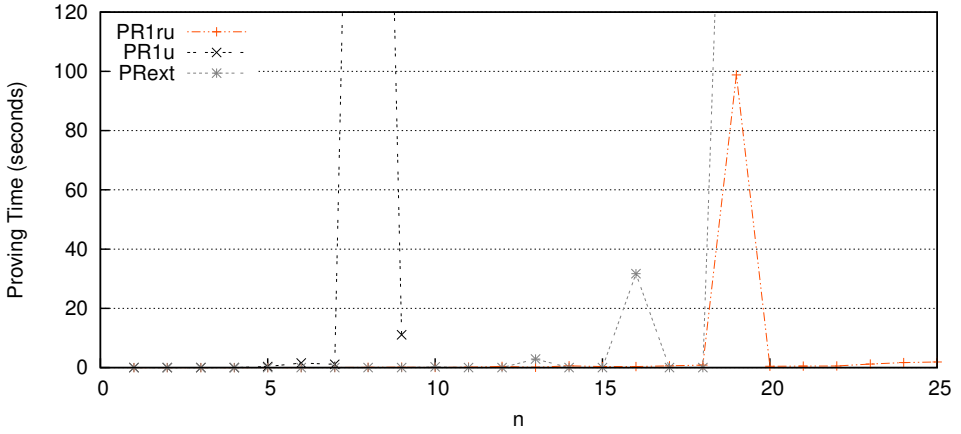


Figure 7.17: Proving time of $T_3(n) \supset q_1$ (random order).

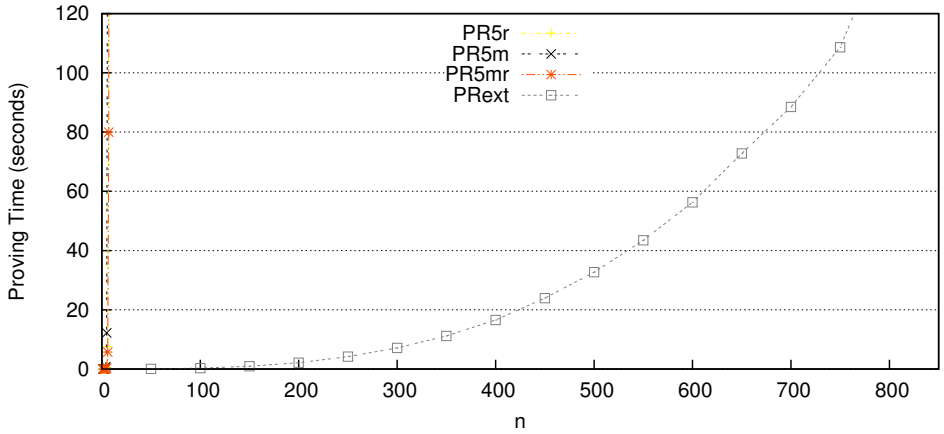


Figure 7.18: Proving time of $T_3(n) \supset p_{\frac{n}{2}+1} \vee q_{\frac{n}{2}+1}$ (sorted order).

Proving $T_3(n) \supset p_{\frac{n}{2}+1} \vee q_{\frac{n}{2}+1}$

The next problem we investigate is to prove $p_{\frac{n}{2}+1} \vee q_{\frac{n}{2}+1}$ from $T_3(n)$. This formula is credulously and skeptically entailed by $T_3(n)$.

Credulous Provers
The backward rule based provers show the same poor performance as for the previous problem while PRext performs very good (cf. Figure 7.18).

The good performance of PRext is not astonishing. All extensions of $T_3(n)$ contain the formula that is to prove and therefore the first encountered extension is a witness for it.

The reason for the poor performance of the backward rule based provers is the same as given for the previous problem.

Skeptical Provers
In the sorted scenario PRext and PR1u perform rather bad while PR1ru performs quite good (cf. Figure 7.19).

PRext has to iterate through all extensions to verify the skeptical entailment. And since there are exponentially many extensions the prover shows the corresponding run time behavior.

As in a previous problem (cf. Figure 7.9) we have for this problem again that use-check is less effective without residual improvement. For this problem the differences are bigger but the reason for that behavior remains the same. By using residual improvement we use use-check information from former proofs that contain less dependencies. Therefore the prover introduces less dependencies and performs better.

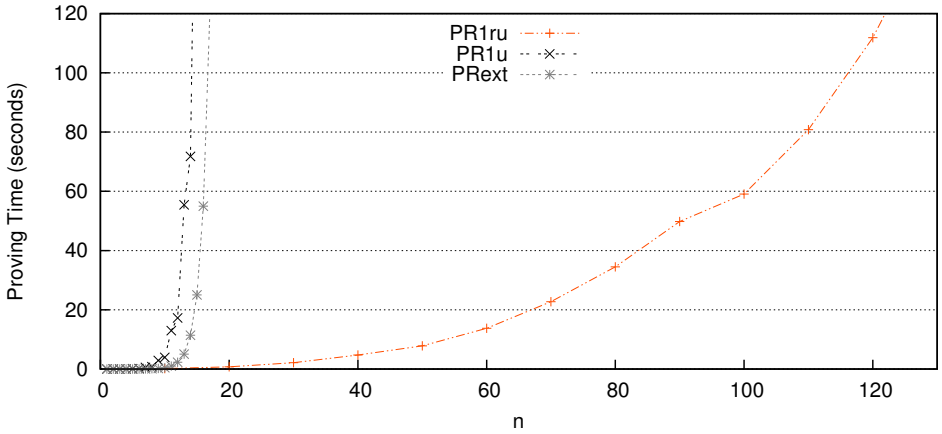


Figure 7.19: Proving time of $T_3(n) \supset p_{\frac{n}{2}+1} \vee q_{\frac{n}{2}+1}$ (sorted order).

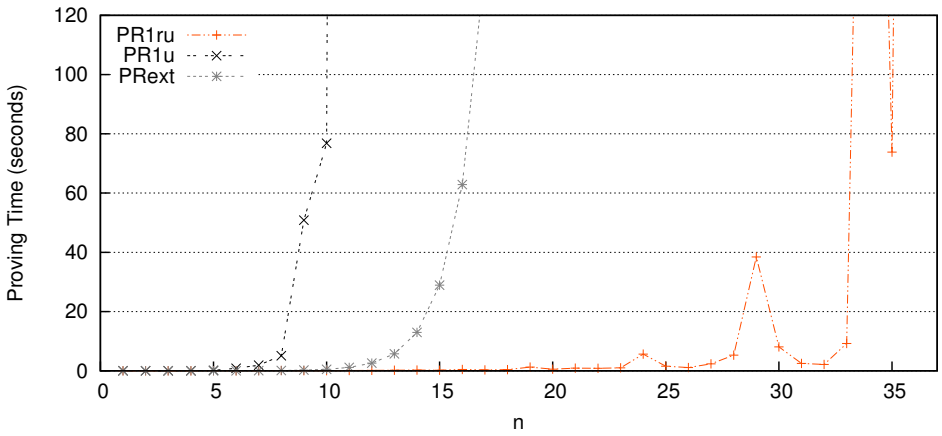


Figure 7.20: Proving time of $T_3(n) \supset p_{\frac{n}{2}+1} \vee q_{\frac{n}{2}+1}$ (random order).

In the random scenario PR1ru drops in performance and shows an unsteady behavior. Nevertheless it is still better than the other two provers which perform as in the sorted scenario (cf. Figure 7.20).

7.4.4 Conclusion

The results for the credulous provers that perform backward rule application are very disappointing. They can only compete with the extension based prover in best case examples. Using the minimal requirements to cut down the proof search tree is not very effective. What the provers miss is a method to predict the failure of a (sD4) application from the information gained in the corresponding failed (sD5) application and to thus cut down the proof

search tree further.

For the skeptical provers that base on partitioning the set of defaults we observer that the strategy to prefer dropping a default does not lead to good results. The provers that follow the oppositional strategy show good performances for some selected problems. Thereby the combination of use-check and residual improvement turns out to be the most promising. Nevertheless the performance of the provers depends highly on the processing order of the residues.
For several other problems that were not discussed here, the partition based provers turn out to be very slow compared to the extension based prover.
One of the reasons for this is that the precondition of a default is not really taken into account by the skeptical default calculus while the extension based provers does. Use-check is a step towards taking the precondition into account, nevertheless its impact is sometimes rather low.
An other reason is the use of an intermediate calculus that handles residues. A more direct approach that bases only on the CPC calculus might lead to better results.

Chapter 8

Conclusion

In this thesis we have successfully developed and implemented several optimization techniques for backward proof search in propositional circumscription and default logic.

As a first result we have encoded use-check information into sequents of classical logic by using labeled formulas and then presented a sound and complete calculus for classical propositional logic in which use-check is encoded. This calculus set the base for our research on circumscription and default logic.

For circumscription we took up the sequent calculus of Bonatti and Olivetti [2] for backward proof search, successfully adapted the idea of use-check to circumscription logic and developed and implemented an algorithm that allows us to omit certain branches in the search tree. As for classical logic we have encoded use-check into circumscription sequents and have presented a sound and complete calculus for propositional circumscription in which use-check is encoded.

Evaluations have shown that backward proof search is superior to use the classical prover together with the syntactic definition of circumscription. In backward proof search both approaches, the one that prioritizes to backward apply non-branching rules first and the one that does it the other way round, showed good results if use-check was involved as an optimization method. The performance of the former approach thereby turned out to be moderately slower but also less influenced by the processing order of the minimized and fixed variables and by the chosen axioms. Our provers were able to compete with mm [22], a minimal model prover that is restricted and thus specialized to sets of clauses.

For the monotonic fragment of justification-free default logic we took up the corresponding sequent calculus of Bonatti and Olivetti [4] for backward proof

search and could slim down the search tree by avoiding several redundancies and by adapting the idea of use-check to it. There we also presented a sound and complete calculus in which use-check is encoded.

Evaluations have shown that the approach to prioritize branching rules in backward proof search is clearly superior to the opposite approach and that use-check is essential for good results.
The prover based on the closure and minimal quasi-supports of the residue theory performed good for most problems. The processing order of the residues turned out to have quite an influence on the performance of the former prover. The latter prover showed a more stable behavior, which rests upon using sorted lists of formulas in the underlying classical prover.

For credulous entailment in default logic we took up the sequent calculus of Bonatti and Olivetti [4] for backward proof search. Since all its deduction rules have only one premise, adapting use-check to it was no option. Instead we presented and implemented a preprocessing step to reduce backtracking and a method to omit proving residue sequents according to cached intermediate results.

For skeptical entailment in default logic we took up the sequent calculus of Bonatti and Olivetti [3] for backward proof search. There we also introduced a method to omit proving residue sequents according to cached intermediate results. Furthermore we successfully adapted the idea of use-check for that prover.

In a further approach we developed an algorithm to compute the extensions of a default theory and implemented it in a way that iterating over the extensions was possible without the need to calculate all extensions beforehand. This extension iterator then served us as a comparison algorithm to the rule based provers.

For credulous entailment the performance of the rule based prover was rather disappointing. The impact of our optimizations were rather small and the extension based provers outperformed these provers by far.
One of the reasons is that the we use a three stage calculus which first reduces the default sequents to residue sequents, then hands over to the residue prover which again hands over to the classical prover. The extension based prover rely solely on the classical prover and follows thus a more direct approach. A further reason is that the rules for proper defaults don't take the prerequisite into account but delegate this to the residue prover.

For skeptical entailment the results were a bit more encouraging. The gap between the rule based and the extension based prover is not so big and for some selected problems the rule based prover performed best. However, in the overall picture the extension based prover perform better.
Although the skeptical calculus is also a three stage calculus and its rules

that process proper defaults also don't take the prerequisite into account it performs clearly better than the corresponding credulous prover. The main difference between the two provers is that we could adapt use-check for the skeptical prover. The assumption that use-check leads to the better performance is therefore nearby. And indeed the performance test indicated that use-check together with the method to reuse intermediate results is the main reason for the reasonable performance.

Further Work

We have observed that the processing order of the minimal and fixed variables and of the defaults and residues can have a big influence on the performance of our provers. It could be worth searching for methods that provide us with good processing orders.

Furthermore we have observed that the strategy to prioritize one rule over the other is sometimes to simple to obtain good results. It would be nice to have a better criteria for the prioritization of a rule.

Then it could be worth developing provers that are tailored to default theories of a certain form. Cholewinski, Marek, Truszczynski and Mikitiuk [8] for example have given a prover for conjunction free default theories that is astonishing fast for some complex problems.

Index

Bibliography

[1] Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. Logics workbench 1.0. In Harrie C. M. de Swart, editor, *TABLEAUX*, volume 1397 of *Lecture Notes in Computer Science*, pages 35–37. Springer, 1998.

[2] Piero A. Bonatti and Nicola Olivetti. A sequent calculus for circumscription. In Mogens Nielsen and Wolfgang Thomas, editors, *CSL*, volume 1414 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 1997.

[3] Piero A. Bonatti and Nicola Olivetti. A sequent calculus for skeptical default logic. In Didier Galmiche, editor, *TABLEAUX*, volume 1227 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 1997.

[4] Piero A. Bonatti and Nicola Olivetti. Sequent calculi for propositional nonmonotonic logics. *ACM Trans. Comput. Log.*, 3(2):226–278, 2002.

[5] Piero Andrea Bonatti. A gentzen system for non-theorems. Technical report, Christian Doppler Labor für Expertensysteme, Technische Universität, Wien, 1993.

[6] Pawel Cholewinski, V. Wiktor Marek, Artur Mikitiuk, and Miroslaw Truszczynski. Experimenting with nonmonotonic reasoning. In *ICLP*, pages 267–281, 1995.

[7] Pawel Cholewinski, V. Wiktor Marek, and Miroslaw Truszczynski. Default reasoning system deres. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *KR*, pages 518–528. Morgan Kaufmann, 1996.

[8] Pawel Cholewinski, V. Wiktor Marek, Miroslaw Truszczynski, and Artur Mikitiuk. Computing with default logic. *Artif. Intell.*, 112(1-2):105–146, 1999.

[9] Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic reasoning: Towards efficient calculi and implementations. In John Alan

Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1241–1354. Elsevier and MIT Press, 2001.

[10] Uwe Egly and Hans Tompits. Proof-complexity results for nonmonotonic reasoning. *ACM Trans. Comput. Log.*, 2(3):340–387, 2001.

[11] Thomas Eiter and Georg Gottlob. Propositional circumscription and extended closed-world reasoning are Π_2^P-complete. *Theoretical Computer Science*, 114(2):231–245, 1993.

[12] David W. Etherington. Relating default logic and circumscription. In John P. McDermott, editor, *IJCAI*, pages 489–494. Morgan Kaufmann, 1987.

[13] Benjamin N. Grosof. Default reasoning as circumscription. In *NMR*, pages 115–124, 1984.

[14] Alain Heuerding, Gerhard Jäger, Michael Schwendimann, and Michael Seyfried. A logics workbench. *AI Commun.*, 9(2):53–58, 1996.

[15] Alain Heuerding, Gerhard Jäger, Stefan Schwendimann, and Michael Seyfried. Propositional logics on the computer. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *TABLEAUX*, volume 918 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 1995.

[16] Tomasz Imielinski. Results on translating defaults to circumscription. *Artif. Intell.*, 32(1):131–146, 1987.

[17] Lefteris M. Kirousis and Phokion G. Kolaitis. A dichotomy in the complexity of propositional circumscription. *Theory Comput. Syst.*, 37(6):695–715, 2004.

[18] Vladimir Lifschitz. Benchmark problems for formal non-monotonic reasoning, version 2.00. In Michael Reinfrank, Johan de Kleer, Matthew L. Ginsberg, and Erik Sandewall, editors, *NMR*, volume 346 of *Lecture Notes in Computer Science*, pages 202–219. Springer, 1988.

[19] Victor W. Marek and Miroslaw Truszczynski. *Nonmonotonic logic - context-dependent reasoning.* Artificial intelligence. Springer, 1993.

[20] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2):27–39, 1980.

[21] John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artif. Intell.*, 28(1):89–116, 1986.

[22] Ilkka Niemelä. A tableau calculus for minimal model reasoning. In Pierangelo Miglioli, Ugo Moscato, Daniele Mundici, and Mario Or-

naghi, editors, *TABLEAUX*, volume 1071 of *Lecture Notes in Computer Science*, pages 278–294. Springer, 1996.

[23] David Poole. A logical framework for default reasoning. *Artif. Intell.*, 36(1):27–47, 1988.

[24] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[25] Raymond Reiter and Giovanni Criscuolo. On interacting defaults. In Patrick J. Hayes, editor, *IJCAI*, pages 270–276. William Kaufmann, 1981.

[26] Alasdair Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1(4):425–467, 1995.