Propositional Logics on the Computer

Alain Heuerding*, Gerhard Jäger, Stefan Schwendimann*, Michael Seyfried

Institut für Informatik und angewandte Mathematik
Universität Bern

Abstract. The purpose of the paper is to present the Logics Workbench, an interactive system aiming to facilitate the access to logical formalisms for nonspecialists as well as specialists. It is an integrated system which provides a library of the most important propositional calculi and many algorithms in this area. Special emphasis is put on a clear design of the human interface and a powerful information system, which covers online help and documentation.

1 Introduction

Our Logics Workbench LWB, which will be presented below, is an interactive system aiming to facilitate the access to and the use of a variety of logical formalisms. So far our focus is on propositional logics but extensions to more general situations are planned for future work.

The LWB is an integrated system which is centered around a comparatively small kernel and which provides separate libraries for the most important propositional calculi. The kernel and the libraries together with further central components of the LWB (see also Fig.1) will be described below.

Since the LWB is not exclusively intended for specialists, a great deal of effort has been spent on a comprehensive information system. This information system is also crucial in connection with one of the main applications of the LWB: It is a useful tool in training and education where it contributes to a better understanding of the interdependencies of logical systems and of the procedural aspects of various logical calculi.

An excuse is in order: It is not possible to cite all the related work and the existing systems in this area. Furthermore, space does not permit to give definitions of the calculi mentioned below. For details and an extensive list of references please refer to the full documentation of the LWB [1].

2 General Aspects

The structure of the LWB is strictly modular. It allows the user to build a personal LWB according to his preferences and interests. It is even possible to

* Work supported by the Swiss National Science Foundation, SPP 5003-34279.

[1] See section 'Access to the LWB'.

link the kernel and selected modules to a third program and to use the LWB in this way as a background logic engine.

Very schematically the LWB comprises four main components, which closely interact with each other: kernel and parser, logic modules, graphical user interface, information system.

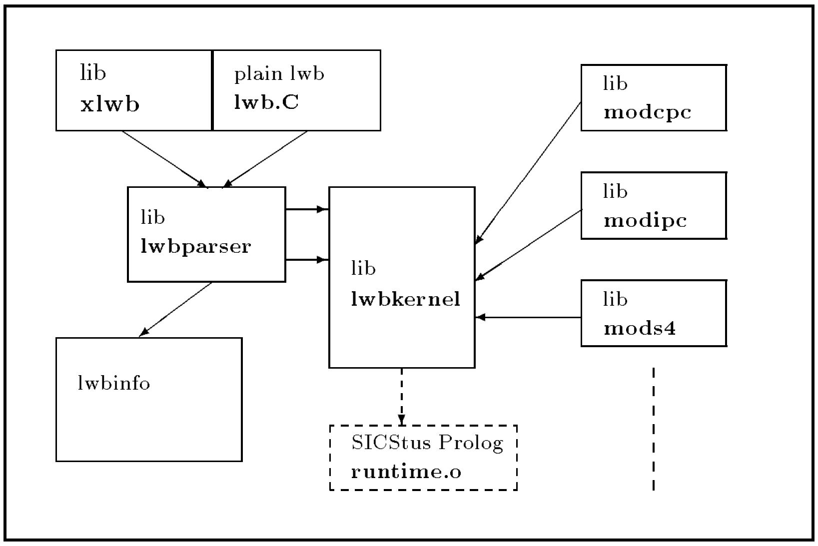

Fig. 1. The modular structure of the LWB.

The LWB kernel provides the central data structures on which both the parser and the logic modules rely. The logic modules, on the other hand, contain a coherent collection of efficient implementations of well-known algorithms of the most important propositional logics:

1. Classical and intuitionistic logic.
2. Modal and multimodal logics.
3. Temporal logics.
4. Nonmonotonic systems.
5. Typed and untyped term calculi (lambda calculus, combinatory logic).

Various features ensure that users are able to interactively explore the system and the contained logic modules without further aids or a lengthy preparation. The basic communication with the system is done in an untyped LWB language, which follows very closely the philosophy of well-established computer algebra systems such as Maple and Mathematica.

A graphical user interface has been developed in order to facilitate the handling of sessions, the navigation within the program, and the re-use of previously typed statements. A comprehensive information system gives detailed support for the usage of the different modules. It will be extended to contain uniformly grouped information and general introductions for selected logics. All information is available via the World Wide Web and may therefore be consulted without any local installations.

3 Kernel and Parser

The propositional formulas and other expressions that are used for algorithms are represented in the kernel of the LWB. The principles of object oriented design have been applied. Great care was taken to provide the developers of LWB modules with a clear, abstracted interface to the basic objects. C++ allows good encapsulation, e.g. pointer handling becomes superfluous for the implementation of algorithms. Moreover, we took advantage of the possibility of operator overloading which makes dealing with expressions more intuitive and increases the readability of functions.

Great importance has been attached to a clear design of the internal class hierarchy and high efficiency of the basic member functions. The addition of new data types is simple; thanks to the inheritance mechanism of classes only those parts that really differ from the other data types need to be implemented. Moreover, there is a close relationship to the parser: the nonterminals of the grammar of the parser correspond to the abstract base classes, and the terminals correspond to the derived concrete classes. Reference counting has been chosen as internal memory management.

To set up a user-friendly parser we used the well-known development tools *lex* and *yacc*. With *lex* the lexical analyzer is generated (recognition of tokens) and with *yacc* the actual parser is built. The description of the accepted language is given in Backus-Naur Form (BNF). Both tools translate the intuitive descriptions into C programs. The parser is designed to respect the usual conventions in the writing of expressions.

A LWB prototype has been written in Prolog, and has later been converted to the C++ environment without difficulties. New algorithms are directly written in C++. However, an interface to SICStus Prolog is integrated and tested, so that users and external contributors may still write algorithms in Prolog. Provided that the SICStus runtime library has been linked to the LWB, such Prolog algorithms can be added to the LWB without changes to the LWB itself.

4 Logic Modules

Each logic module provides a series of auxiliary functions (e.g. length and depth of formulas or proofs) and standard functions for the computation of various normal forms and the like. In addition, there are tools to compare different calculi and to compute embeddings and reductions.

We have the possibility to generate 'random formulas', depending on given parameters like depth or number of literals. Of course the notion of random formula is problematic but there are approximations that are sufficient for our purpose (cf. the generation of random polynomials). These random formulas are used in order to study the efficiency of our implementations, especially with respect to average case behaviour.

In order to structure the description of the logic modules, we distinguish between monotonic propositional systems, nonmonotonic propositional systems and the Curry-Howard module.

4.1 Monotonic Propositional Systems

The largest group of implemented modules concerns monotonic propositional logics. The LWB tries to cover a great variety of propositional logics (including the most important ones), but it is not our aim to include every system which is described in the literature.

In this paper it is not possible to describe each of the logics which have been implemented or each of the methods which have been used in order to achieve the corresponding implementation. The following list just names the monotonic systems which are presently treated by the LWB.

1. Classical and intuitionistic propositional logic.
2. Modal propositional logics K, KT, $\mathsf{S4}$, $\mathsf{S5}$.
3. Provability logic.
4. Multimodal logics KT_n, $\mathsf{S4}_n$, $\mathsf{S5}_n$.
5. Tense logics.
6. Linear Logic.

For our implementation of intuitionistic propositional logic we have partly made use of Dyckhoff [3] and Hudelmaier [8]. In the presentation of the modal propositional logics and of provability logic we follow Fitting [4, 5] and Boolos [2]. Our approach to multimodal logics is similar to the presentation in Halpern and Moses [6]. Tense logics are implemented via a calculus developed by Zimmermann [14], which is based on Rescher and Urquhart [10]. In the case of linear logic we follow Andreoli [1] and Tammet [13].

Decision Procedures: All decision procedures for monotonic systems, which are implemented so far, are based on sequent calculi. We took the standard forms of these systems and combined them with several methods that improve the runtime complexity considerably.

For each logic under consideration a specifically tailored decision procedure is used. We do not follow the suggestion of a number of authors to treat several logical calculi simultaneously by reducing them to one specific base system.

For example, there exists a straightforward embedding of the modal logic KT into K. Although the embedding is very simple, nested modal operators and equivalences can cause an exponential increase of the length of the formula. If one uses structure sharing and if no equivalences occur in the formula, the problem can be solved. However, proof search in K for the embedded formula is still considerably harder than proof search for the original formula in KT, because

KT allows a much stronger form of use-checking (see section 'Efficiency') than K.

Besides pure derivability, the LWB is also able to cope with provability of a formula in a logic plus theory, i.e. in a logic plus additional nonlogical axioms. Of course this is trivial for classical and intuitionistic logic, but requires some effort in the case of several modal logics. Two examples are the wise-men-problem in multimodal logic and the proof of the equivalence of different modal logics (e.g. S5 and KT5).

Simplification of Formulas: It is not difficult to give some definition of 'simplification' with nice properties. Two possibilities are the conjunctive normal form or the minimized conjunctive normal form. In a few cases such normal forms of this kind may help, but in general the results are far too long. Moreover, we think that the user should be able to recognize the structure of his input formula after the simplification process – a condition that is often ignored by transformations to normal forms.

Therefore, we sacrifice the requirement to obtain a 'minimal' solution in some theoretical sense. Simplification of a formula can be vaguely described in our context as the transformation of a formula to a logically equivalent form which is better readable, yet does not destroy the original structure unnecessarily.

These requirements called for the development of a rather sophisticated algorithm. On the one hand it tries to find as many simplifying sequences of transformations as possible. On the other hand, for efficiency reasons, it cannot use an exhaustive search to find these sequences, but relies on a very directed search. In addition the user may control (via parameters) the tradeoff between quality and runtime of the simplification process according to his needs.

It is complicated to exactly define the specifications of such simplification functions and to implement them efficiently; nevertheless their presence is convenient for the user (cf. their usage in computational algebra).

In order to facilitate queries to the LWB, we do not require the input being in normal form. Even a seemingly simple transformation such as the step from $A \leftrightarrow B$ to $(A \rightarrow B) \wedge (B \rightarrow A)$ can convert a well structured formula into a completely unreadable one.

Efficiency: We use several methods to increase the efficiency of the implemented algorithms. None of the methods described below is completely new. However, it turned out that their combination considerably improves the runtime behaviour without leading to unintelligible algorithms.

Structure sharing is used on several occasions. On the one hand it is supported by the kernel of the LWB: If the user types `a := p v q & r;` and afterwards `b := a v p;`, then the formula `a` is not copied in order to store `b`, but simply a pointer to `a` is used. This method helps to save memory and can speed up equality tests. On the other hand most decision procedures make use of structure sharing in order to avoid the copying of the whole sequent when only the head of the sequent changes.

Use-checking is a widely applicable method in the context of proof search for sequent calculi (cf. Sahlin, Franzén and Haridi [11]). The basic idea is the following: Suppose we have to check whether $A \wedge B, \Gamma$ is provable. First we check whether A, Γ is provable. If this is the case and if A was not 'used' in the proof of A, Γ, then it is superfluous to search for a proof of B, Γ since the provability of this sequent follows immediately from that of A, Γ.

Of course the definition of 'used' depends on the logic under consideration. For example, provability logic and some modal logics (e.g. KT, S4 and S5) require different definitions. Already in classical propositional logic use-checking usually cuts off many branches from the search tree (consider for example the pigeonhole formulas). The effect is even stronger in the case of modal and multimodal logics plus theory, where the same formula can cause unnecessary branchings over and over again.

Another method is the *efficient loop-checking* for the modal logic S4. Because the usual sequent calculus for S4 contains a hidden contraction, loops can occur during proof search. Therefore we have to test for each sequent whether it already occurred on the current branch of the search tree. Such a loop-check is in general expensive, and the implementation of an equality test for sequents is laborious. In the case of S4 there exists a calculus where loop-checking is very cheap: Only when a ($\Box$) rule is applied backwards, we have to check whether the main formula is member of a list of formulas. The length of this list is at most the number of previous reverse applications of the ($\Box$) rule on this branch, but is in general much shorter. (A simpler version of this calculus can be found in Ladner [9].) The main problem, namely the presence of noninvertible rules, still remains, but this problem is related to the PSPACE property of decision procedures for S4 and therefore difficult to avoid.

4.2 Nonmonotonic Propositional Systems

The LWB contains the propositional variants of the nonmonotonic systems Closed World Assumption, Circumscription, Default Logic and Autoepistemic Logic.

Most of the interesting properties as well as the known problems inherent in the systems above do already show up in the propositional case, and it is sometimes even easier in this framework to construct examples which illustrate these problems. Therefore, at least for didactical purposes the restriction to the propositional case should not be too disturbing.

Following the pattern of monotonic systems a number of functions besides decision procedures have been implemented. A good example is the function that computes the set of extensions of a default theory. This function gives a lot more insight into Default Logic than the mere provability test.

4.3 Curry-Howard Module

The Curry-Howard module extends the functionality of the LWB by providing two components: (i) an environment for interactive proof construction in intuitionistic logic and related nonclassical (resource-bounded) logics (minimal logic, relevance logic) and (ii) a platform, which allows computational experiments with terms from lambda calculus and combinatory logic. The logical formalisms and the term calculi are tightly connected via the Curry-Howard isomorphism, see for example Hindley and Seldin [7].

For both term systems, lambda calculus and combinatory logic, we have implemented the standard functions, such as substitution operations using alpha-conversion, contraction of redices and normalization of terms. We have strictly adhered to the leftmost-outermost strategy, which guarantees termination of the normalization process for all terms that possess a normal form, in fact normal forms are computed by repeated contraction of the LO-redex in terms. Wherever appropriate we have specified the algorithms for typed and untyped terms of both calculi. In addition there exist transformations between the different term calculi, including the star-translation of Curry and Feys and the 'forgetful' interpretation, as well as functions mapping terms of both lambda calculus and combinatory logic to formulas of intuitionistic (minimal) logic, such as computation of principal pairs and principal type schemes. Convenience of term input and readability of output are augmented by allowing special formats for in- and output of nested lambda abstractions.

The 'logical' part of the Curry-Howard module consists of the functions that allow to construct intuitionistic (minimal, relevant) proofs in a natural deduction style and a proof search algorithm for the fragment of intuitionistic logic containing implication and conjunction. This algorithm can be naturally derived from a purely syntactic modification of Kleene's well-known sequent calculus for intuitionistic logic (see Seyfried [12]). This modification is characterized by the restriction of the applicability of some rules of the original calculus, yet retaining logical completeness and various other 'natural' properties of Kleene's sequent calculus.

Let us briefly sketch the main idea: We consider sequents of the form $\Gamma \supset A$. The axioms of our calculus are

$$\Gamma, A \supset A$$

and the two rules for implication are

$$\frac{\Gamma, A \supset B}{\Gamma \supset A \to B}(\to R) \qquad \frac{\Gamma, A \to B \supset A \qquad \Gamma, A \to B, B \supset P}{\Gamma, A \to B \supset P}(\to L)$$

where $(\to L)$ is only applicable, if the head $\mathsf{h}(A \to B)$ of $A \to B$ is the atomic formula P. In this context the head $\mathsf{h}(F)$ of an implicational formula F is inductively defined as follows: $\mathsf{h}(P) := P$ if P is atomic; $\mathsf{h}(F_0 \to F_1) := \mathsf{h}(F_1)$.

The computationally significant achievement of the derived decision method is its efficiency (loop-checking is done as for S4), being far superior to the original algorithm in the average case. Thus the calculus is a step towards combining the virtues of Kleene's calculus with the advantage of modern calculi for intuitionistic logic, whose primary design goal is efficiency of the inherent decision algorithms.

Since the constructed and/or generated proofs are internally represented as lambda terms, a function has been added to the Curry-Howard module, which allows to produce TEX-source from the lambda terms, thus generating a presentation which is accessible to non-specialists as well as aesthetically more appealing than proof-terms.

5 Graphical User Interface (GUI)

The GUI has been developed under *OSF/Motif Release 1.2.2. Motif* has become a widely accepted standard and so the future support and the portability to other platforms are guaranteed. The GUI contains 3 components (see also Fig.2):

1. The main window: The main window consists of a menu bar and a scrollable input window, which can be changed in size. The input window is the essential part of the GUI. It is built up by pairs of input and output regions. By returning to previous input regions the user can save repeated typing of equal or similar commands. Each input region is a little multiline text editor, in which all needed text manipulations can be done.
2. Status window: It displays general information about the state and the activity of the LWB and specific information about the run of algorithms, which is of interest to the user. The amount of information can be controlled.
3. Option window: In the option window the user can change global variables like display mode or bracket mode by mouse clicks. Likewise he can choose his favourite colour settings.

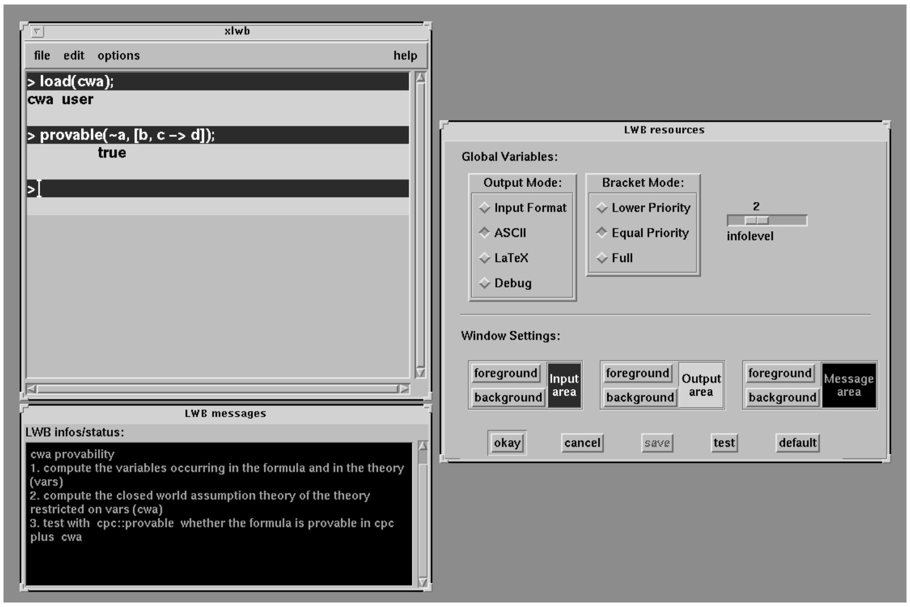

Fig. 2. The graphical user interface.

Special emphasis has been put on the similarity to other X11-/Motif-applications. For example, user specific adaptions are done in the usual way (resource file). A fully working first version supports various useful features:

1. Save/Load of all assigned variables, or the list of input and output regions (session).
2. Insertion and deletion of regions. Hiding the output regions. Reevaluation of all input regions.
3. Emacs-like control sequences.
4. Connection to the help system (via help button).

6 LWB Information System

The LWB information system goes far beyond what is usually offered by other systems of computer assisted logics and mathematics.

It relies on the features provided by World Wide Web (WWW). This approach offers several advantages:

1. Representation: The html format includes fonts in different shapes and sizes, but also icons and pictures.
2. Hypertext links: The LWB information system uses the capabilities of hypertext to structure the available information.
3. Availability: Any WWW Browser (e.g. Mosaic) is sufficient to use the LWB information system installed in Berne. Users from outside Berne can also install it in a comfortable way at their site.
4. Flexibility: A key feature is the easy extendability. Existing parts are not affected by changes or extensions.

The LWB information system also contains a large variety of information besides the standard help texts:

1. Information about the LWB: Aims, news, planned work, installation instructions.
2. Sample session with the LWB, picture of a sample session with the XLWB.
3. Syntax description.
4. Description of each function (with examples), list of the functions in a module, list of the modules.
5. Reference manual (Postscript).

Moreover, it is possible to use the LWB-installation in Berne via the information system. This facility is especially useful for occasional queries, because no preceding local installation of the LWB or of the LWB information system is required. Naturally this usage of the LWB is not very user-friendly, and the number of simultaneous users is limited.

The reference manual contains a chapter that describes both theory and implementation for each of the implemented logics. The theory section contains the basic calculi, definitions and theorems, illustrated and motivated by typical examples. In order to explain the chosen algorithms, several implementation-oriented variants of the calculi are introduced. For the same reason some definitions and theorems are reformulated.

The function descriptions of the LWB information system are automatically generated from the corresponding description in the reference manual, thus ensuring consistency. Moreover, all examples in the reference manual are automatically calculated by the LWB during typesetting and the results are inserted. Therefore, the reader cannot be confused by examples that do not correspond to the behaviour of the LWB.

7 Tests

The Logics Workbench has been thoroughly tested. For this purpose we have compiled extensive lists of test formulas, among them most standard examples from the literature. Parameterized test formulas (e.g. the pigeonhole formulas) are especially helpful in comparing different methods for various tasks (e.g. provability tests, normal forms). However, many of these enjoy a strong regular structure, so that estimates about the computational behaviour of algorithms have to be taken with care. Small changes of the implementation can alter the runtime dramatically. Moreover, in the case of decision procedures for non-classical logics, most of the examples that can be found in the literature have toy character, and the formulas obtained via embeddings cover only certain cases.

The above considerations led us to include random formulas in the test phase. However, the direct use of random formulas is often impossible, consider for example the case of decision procedures: Computing the correct result by hand is too time-consuming, while randomly generated formulas are likely to be unprovable. Hence the construction of a priori provable formulas is essential, one can take a randomly generated formula A and compute a negation normal form B of A; then the formula $A \leftrightarrow B$ is provable, but the decision procedure cannot 'see' this fact without proof search (naturally no pre-simplification must take place). A beneficial side-effect of the search for 'real-world' examples is that they give valuable information about the average case behaviour of algorithms, whereas in the literature in general only the worst case behaviour is discussed.

8 Work in Progress

8.1 Programming Language

A simple imperative LWB programming language is under development, and currently the following features are included: procedure declaration with call-by-value parameters and local variables, creation of selected data structures within procedures, a simple if-statement, calls to other procedures, recursion. The programming language will later be integrated in the LWB.

8.2 Extension of the LWB Information System

The LWB information system will be extended by background information about the implemented logics. This information is already partially contained in the reference manual, but the new representation will be much more user-friendly. It will offer facilities like hypertext links (instead of mere page numbers) and search queries. For some selected logics also tutorials will be available.

8.3 New Modules and Algorithms

The following logics modules and algorithms are in the process of being implemented: multivalued logics, temporal logics, second-order propositional logic, proof construction for nonclassical logics.

9 The LWB in Use

The LWB is used at the University of Berne in connection with courses on logic and computer science. It has turned out to be extremely helpful in considerably reducing the time spent on tedious calculations and transformations that do not give much insight, so that more time was left to concentrate on the essential problems.

Using the LWB it is possible to discuss problems of realistic size which go far beyond the usual toy examples. In this way students get a better feeling for algorithmic aspects of logic in general and especially for the complexities (run time, length of formulas, ...) relevant in practice.

10 Access to the LWB

The LWB can be used via a World Wide Web Browser (e.g. Mosaic), i.e. without preceding installation of any part of the LWB. Just load the LWB home page (address: `http://lwbwww.unibe.ch:8080/LWBinfo.html`) and choose the item `run a session`. Afterwards one can type in the input in a window and submit it. A computer at the IAM in Berne calculates the results and sends them back. The number of simultaneous users and the CPU time per job is limited.

Also the newest version of the reference manual is available via WWW and the LWB home page.

11 Acknowledgements

Special thanks go to several people who have contributed to the LWB: F. Achermann, P. Balsiger, M. Bianchi, P. Brambilla, K. Guggisberg, G. Liechti, H. Zimmermann.

Appendix

In this appendix we present two sample sessions, one based on monotonic systems and the other on nonmonotonic systems. The picture below shows a snapshot of the graphical user interface.

```
   LWB  -  The Logics Workbench

> a := (~(p v q) & (s <-> r) -> q v p)
>      & (((d -> c) -> d) -> d);
```

We have a look at the properties of this 'arbitrary' formula in classical propositional logic (`cpc`*).*

```
> load(cpc);
   cpc user
cpc> b := simplify(a);
   b := (s <-> r) -> q v p
```

Now let us look at the formula `a` *again. It is a conjunction, and the second conjunct has the form*

```
cpc> a2 := a[2];
   a2 := ((d -> c) -> d) -> d
```

Now we compare `a2` *with the formula of Pierce ...*

```
cpc> fml("pierce");
   ((p1 -> p2) -> p1) -> p1
```

... and realize that they are identical modulo renaming. Thus `a2` *must be provable in classical, but not in intuitionistic logic (*`ipc`*).*

```
cpc> load(ipc);
   ipc cpc user
ipc> cpc::provable(a2);
   true
ipc> provable(a2);
   false
```

Now we embed `a2` *in the modal logic* `s4`*. The result must not be provable in* `s4`*, however, it is provable in* `s5`*.*

```
ipc> load(s4, s5);
   s5 s4 ipc cpc user
ipc> ipc::s4(a2);
```

```
   box(box(box(box d -> box c) -> box d) -> box d)
s5> s4::provable(ipc::s4(a2));
   false
s5> s5::provable(ipc::s4(a2));
   true
```

The second example shows a typical difference between Closed World Assumption (`cwa`*) and Default Logic (*`default`*). The theory* `t` *contains statements about the colours of a cube and a ball, namely: The ball or the cube is red, both ball and cube are either red or blue. We want to minimize the number of red objects. The Closed World Assumption theory of* `t` *restricted on* `red(ball)` *and* `red(cube)` *is inconsistent because it contains both* `~red(ball)` *and* `~red(cube)`*.*

```
> t := [red(ball) v red(cube),
>       red(ball) <-> ~blue(ball), red(cube) <-> ~blue(cube)];
load(cwa);
   cwa user
cwa> consistent(t, [red(ball), red(cube)]);
   false
cwa> cwa(t, [red(ball), red(cube)]);
   [red(ball) v red(cube),
    red(ball) <-> ~blue(ball), red(cube) <-> ~blue(cube),
    ~red(ball), ~red(cube)]
```

In Default Logic, we can express the minimality of red objects with the two rules in `Delta`*. The first rule* `[true, ~red(ball), ~red(ball)]` *says: If we can add* `~red(ball)` *to the theory without obtaining an inconsistent theory, then we add* `~red(ball)`*. The second rule states the same for* `~red(cube)`*.*

We obtain two consistent extensions, one contains `~red(ball)` *and the other one contains* `~red(cube)`*.*

```
cwa> load(default);
   default cwa user
default> Delta := [ [true, ~red(ball), ~red(ball)],
default>            [true, ~red(cube), ~red(cube)] ];
default> extensions([t, Delta]);
   [ [red(ball) v red(cube),
      red(ball) <-> ~blue(ball), red(cube) <-> ~blue(cube),
      ~red(ball)],
     [red(ball) v red(cube),
      red(ball) <-> ~blue(ball), red(cube) <-> ~blue(cube),
      ~red(cube)] ]
```

References

1. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.
2. George Boolos. Provability, truth, and modal logic. *Journal of Philosophical Logic*, 9:1–7, 1980.
3. Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *The Journal of Symbolic Logic*, 57(3):795–807, 1992.
4. Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics.* Reidel, Dordrecht, 1983.
5. Melvin Fitting. First-order modal tableaux. *Journal of Automated Reasoning*, 4:191–213, 1988.
6. Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
7. Roger J. Hindley and Jonathan R. Seldin. *Introduction to Combinators and Lambda Calculus.* Cambridge University Press, 1986.
8. Jörg Hudelmaier. An $O(n \log n)$-space decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1):63–75, 1993.
9. Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM journal on computing*, 6(3):467–480, 1977.
10. Nicholas Rescher and Alasdair Urquhart. *Temporal Logic.* Springer, 1971.
11. Dan Sahlin, Torkel Franzén, and Seif Haridi. An intuitionistic predicate logic theorem prover. *Journal of Logic and Computation*, 2(5):619–656, 1992.
12. Michael Seyfried. A sequent calculus for proof search in the $(\rightarrow, \wedge)$-fragment of intuitionistic logic. Technical notes, 1994.
13. Tanel Tammet. Proof search strategies in linear logic. Programming Methodology Group report 70, Department of Computer Sciences, Chalmers University of Technology, Göteborg, Sweden, 1993.
14. Heinrich Zimmermann. A directed tree calculus for minimal tense logic. Master's thesis, IAM, University of Bern, Switzerland, 1994.