The Logics Workbench LWB: A Snapshot

Alain Heuerding[1], Gerhard Jäger, Stefan Schwendimann[1], Michael Seyfried
Institut für Informatik und angewandte Mathematik
Universität Bern, Switzerland

Abstract

The Logics Workbench is an interactive system which gives a survey of a broad variety of propositional logics. One application is as a supporting tool for teaching activities at various levels, but it also offers efficient algorithms for experts. The carefully designed human interface comes with a powerful information system. Therefore it should be easy also for non-specialists to use the LWB with profit.

1 Introduction

This article aims on giving a short overview of the Logics Workbench LWB, which is presently being developed at our institute. It is work in progress so that we confine ourselves to the description of principle design decisions and to those parts of the LWB which – most likely – will not be affected by future efforts.

The LWB 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.

The LWB is used 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 is 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.

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 (see section 8).

[1]Work supported by the Swiss National Science Foundation, SPP 5003-34279.

2 The General Design of the LWB

The structure of the LWB is strictly modular. It even allows the user to build a personal version of the LWB according to his preferences and interests. Moreover, it is possible to 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, parser & interpreter, logic modules, graphical user interface, information system.

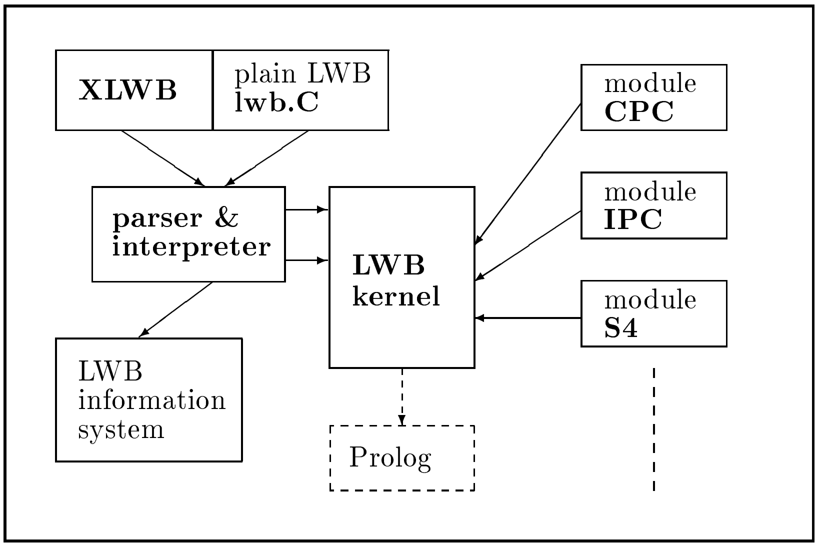

Figure 1: The modular structure of the LWB.

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:

- Classical and Intuitionistic Logic.
- Modal and multimodal logics.
- Temporal logics.
- Nonmonotonic systems.
- 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 reuse 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, Parser & Interpreter

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. Reference counting has been chosen as internal memory management method.

An interpreted Pascal-like programming language is part of the parser. It supports nested procedures, lexical scoping, call by value and call by reference, and provides access to the most important internal data structures such as linked lists and arrays. It allows the user to adapt the LWB according to his preferences and to extend it with own functions.

Connection to Prolog: An interface to SICStus Prolog is integrated so external contributors may still write algorithms in Prolog, which is widely established in the theorem proving community.

4 Logic Modules

Each logic module provides a series of functions for dealing with logical systems. In order to structure the description of the logic modules, we distinguish between monotonic propositional systems, non-monotonic 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.

- Classical and intuitionistic propositional logic.

- Modal propositional logics K, KT, $\mathsf{S4}$, $\mathsf{S5}$.
- Provability Logic.
- Multimodal logics KT_n, $\mathsf{S4}_n$, $\mathsf{S5}_n$.
- Tense logics.
- Linear logic.

Intuitionistic logic is the kind of logic needed in the context of constructive mathematics. An implication $a \rightarrow b$ is often interpreted procedurally as a transformation of (a proof of) a into (a proof of) b. The modal logics allow to express modalities as necessity and possibility or knowledge and belief; $\Box a \rightarrow \Box\Box a$, for example, can be read as "if we know a then we know that we know a". Tense logics provide operators to talk about the future and the past; $\mathsf{GF}a$ means "for all (G) moments in the future there is (F) a moment in the future where a". In Linear Logic formulas on the left of an implication are deemed resources; hence $a \rightarrow (a \wedge a)$ (in Linear Logic syntax: $a \multimap (a \otimes a)$) does not hold — the availability of a does not imply the availability of twice a.

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.

Simplification of Formulas: Simplification of formulas is a standard topic in logic. In our approach to this problem we sacrifice the requirement to obtain a 'minimal' solution in some theoretical sense. LWB simplification of a formula can be vaguely described 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.

Embeddings, Normal Forms, Random Formulas: Besides decision procedures and simplification functions there are functions to compute embeddings and normal forms. We also 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.

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 basically means that multiple occurrences of formulas or subformulas are dealt with suitable pointer techniques. It helps to save memory and can speed up equality tests.

Most decision procedures base on a depth-first search in a tree. One method to improve the efficiency of the search is the so-called use-checking. It is monitored whether the parts which distinguishes the branches from their neighbours are relevant ('used'). If they are not, the neighbouring branches are skipped. This use-checking can efficiently be implemented and leads often to considerable speed-ups.

For some logics, the search tree is not finite but it is known that only finitely many different nodes can occur which implies that the search will eventually loop on every branch. Since complete state comparisons for loop detection are expensive it is worthwile to carefully analyze the conditions under which such loops can occur in a specific logic.

4.2 Nonmonotonic Propositional Systems

Non-monotonic logics are an important topic in artificial intelligence and knowledge representation frameworks. In contrast to the logical systems mentioned before, in non-monotonic systems we can (or must) revise previously deduced beliefs if we learn of new facts. Therefore it is often used if one has to deal with incomplete data or if one has to process incomplete and uncertain information.

The following trivial example shows how non-monotonicity is handled in Default Logic. The theory `t` contains facts 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. Moreover we assume that in general objects are not red. With this assumption we could deduce naively that neither the ball nor the cube is red. This leads to an inconsistency, since we know that at least one of them is red (first formula in the theory `t`).

In Default Logic, we use two rules to formalize our assumption that in general objects are not red. The first rule `[true, ~red(ball), ~red(ball)]` says: If we can add `~red(ball)` to the theory without obtaining an inconsistency, then we add `~red(ball)`. The second rule states the same for `~red(cube)`. If we apply these rules, we obtain two so-called extensions of the theory. Because of the cautious procedure both extensions are consistent, but we cannot decide without additional facts which extension is 'the right one'. Note that we use only a very small part of the possibilities of Default Logic in this example.

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

$$[[\mathit{red}(\mathit{ball}) \lor \mathit{red}(\mathit{cube}),\ \mathit{red}(\mathit{ball}) \leftrightarrow \neg \mathit{blue}(\mathit{ball}),\ \mathit{red}(\mathit{cube}) \leftrightarrow \neg \mathit{blue}(\mathit{cube}),\ \neg \mathit{red}(\mathit{ball})]$$
$$[\mathit{red}(\mathit{ball}) \lor \mathit{red}(\mathit{cube}),\ \mathit{red}(\mathit{ball}) \leftrightarrow \neg \mathit{blue}(\mathit{ball}),\ \mathit{red}(\mathit{cube}) \leftrightarrow \neg \mathit{blue}(\mathit{cube}),\ \neg \mathit{red}(\mathit{cube})]]$$

Besides Default Logic, also Circumscription, Closed World Assumption, and Autoepistemic Logic are implemented in the LWB.

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.

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. 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.

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):

- 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.

- 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.
- 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.

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:

- Save/Load of sessions and kernel states.
- Insertion and deletion of regions. Hiding the output regions. Reevaluation of all input regions.
- Emacs-like control sequences.
- Connection to the information system (via help button).
- Name completion. Parentheses highlighting.

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:

- Representation: The HTML format includes fonts in different shapes and sizes, but also icons and pictures.
- Hypertext links: The LWB information system uses the capabilities of hypertext to structure the available information.
- Availability: Any WWW Browser (e.g. Mosaic or Netscape) 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.

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

- Information about the LWB: Aims, news, planned work, installation instructions.
- Sample session with the LWB.
- Description of the syntax and the programming language.
- Description of each function (with examples), list of the functions in a module, list of the modules.
- Reference manual.
- Theoretical background for some of the implemented logics.

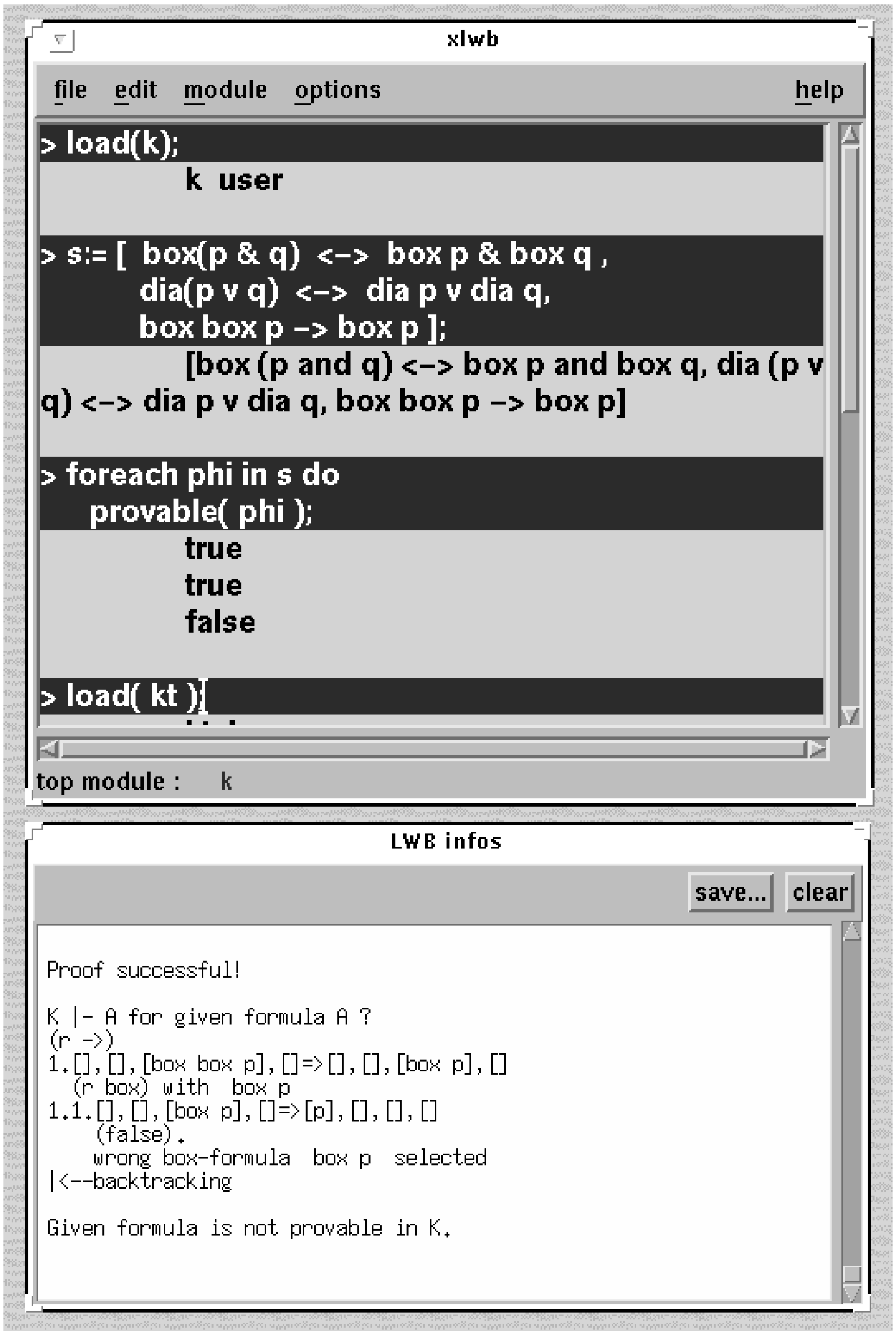

Figure 2: The graphical user interface.

7 Tests

The Logics Workbench has been thoroughly tested. For this purpose we have compiled extensive lists of test formulas. We obtained these test formulas from different sources, each having its drawback.

Of course we considered the relevant literature. However, especially in the case of decision procedures for non-classical logics, most examples that can be found in the literature have toy character. Exceptions are parameterized test formulas (e.g. the pigeonhole formula) that can be made arbitrarily hard. However, using only such formulas for tests is dangerous: Because of the regular structure of these formulas some parts of the algorithm are perhaps never tested. Using formulas obtained via embeddings has the same disadvantage.

In some cases we succeded in applying theoretical results about the logics (e.g. non-derivability theorems in modal logics) to obtain additional hard formulas.

Another source for test examples were random formulas. However, the direct use of such formulas is often impossible, since computing the correct result by hand is too time-consuming. Hence the construction of a priori provable formulas is essential.

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 Remote Access via Internet

The LWB can be used via a World Wide Web Browser (e.g. Mosaic or Netscape), i.e. without preceding installation of any part of the LWB. This facility is especially convenient for occasional queries. `http://lwbwww.unibe.ch:8080/LWBinfo.html` is the address of the LWB home page. Just load it 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.

Appendix

The following example involves three monotonic logics, namely classical, intuitionistic and linear logic. It is to show how the logics can be compared by using embeddings.

We start off by loading the module for classical propositional calculus.

```
> load( cpc );         # classical propositional calculus
        cpc  user
```

We get a predefined formula.

```
> s:= urquhart( 2 );
```

$$((p1 \leftrightarrow p2) \leftrightarrow p1) \leftrightarrow p2$$

For better readability we substitute a for $p1$ and b for $p2$;

```
 > s1:= s { a/p1, b/p2 };
```

$$((a \leftrightarrow b) \leftrightarrow a) \leftrightarrow b$$

From now on we omit unnecessary parentheses.

```
 > set("bracketmode","assoc");
```

We load the module for intuitionistic propositional calculus and check the provability in both logics.

```
> load(ipc);        # intuitionistic prop. calculus
       ipc  cpc  user
> cpc::provable( s1 );  ipc::provable( s1 );
```

$$\mathit{true}$$
$$\mathit{false}$$

This is due to the fact that while the commutativity of '$\leftrightarrow$' holds in `ipc` as well, ...

```
> ipc::provable( (a <-> b) <-> (b <-> a) );
```

$$\mathit{true}$$

... the associativity of '$\leftrightarrow$' does not:

```
 > ipc::provable( (a <-> (b <-> c) )  <->  ((a <-> b) <-> c)  );
```

$$\mathit{false}$$

We embed `s1` in `ipc` in two different ways:

```
> s2:= cpc::ipc( s1 );  cpc::ipc( s1, "goedel" );
```

$$\neg\neg(a \leftrightarrow b \leftrightarrow a \leftrightarrow b)$$
$$\neg\neg a \leftrightarrow \neg\neg b \leftrightarrow \neg\neg a \leftrightarrow \neg\neg b$$

The embedded formula is of course provable.

```
 > ipc::provable( s2 );
```

$$\mathit{true}$$

We load the module for propositional linear logic, for which a special parser mode is triggered to recognize the linear operators.

```
> load( ll );   # propositional linear logic
        ll  ipc  cpc  user
Entering special mode for linear logic.
```

We embed the intuitionistic formula into `ll` :

```
 > s3:= ipc::ll( s2 );
```

$$!(!((!((!((!a \multimap b) \,\&\, (!b \multimap a)) \multimap a) \,\&\, (!a \multimap (!a \multimap b) \,\&\, (!b \multimap a))) \multimap b) \,\&\, (!b \multimap (!((!a \multimap b) \,\&\, (!b \multimap a)) \multimap a) \,\&\, (!a \multimap (!a \multimap b) \,\&\, (!b \multimap a)))) \multimap 0) \multimap 0$$

Full linear logic is undecidable, so increase a measure of depth to prove the embedded formula. `> for d:= 3 to 5 do ll::provable( s3 , d);`

$$\mathit{fail}$$
$$\mathit{fail}$$
$$\mathit{true}$$

9 Acknowledgements

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