University of St Andrews

Full metadata for this thesis is available in
St Andrews Research Repository
at:
http://research-repository.st-andrews.ac.uk/

This thesis is protected by original copyright

A PROOF EDITOR FOR LINEAR LOGIC

THOMAS PEILLON

DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL
SCIENCES
THE UNIVERSITY OF ST ANDREWS

BIBLIOTHECA UNIVERSITATIS ANDREANA

Abstract

This thesis describes the design and implementation of an interactive proof editor for Linear Logic. The proof editor is based on the MacLogic proof editor developed by the MALT project, at the University of St Andrews.

We describe how the proof tactics can be derived from the sequent calculus rules for Intuitionistic and Classical Linear Logic and how these are encoded. We show why transformations need to be applied to formulae during the proof process, and how these are implemented.

We also describe theorem provers for Classical and Intuitionistic Linear Logic, which are used to guide the user through the proof process, by warning of attempts to solve problems that the theorem prover has been unable to solve.

I hereby declare that the research described in this thesis was carried out by myself, that the thesis was composed by myself, and that the thesis has not been accepted in partial or complete fulfilment of any other degree or professional qualification.

T. Peillon

Acknowledgements

I would like to thank my supervisor Roy Dyckhoff for his supervision and guidance, as well as access to the MacLogic source code, parts of which were used in this project.

I would also like to thank the Department of Mathematical and Computational Sciences for providing a lively and stimulating research environment, and my family for their support during the last year.

TABLE OF CONTENTS

1 LINEAR LOGIC

2 THE PROOF EDITOR

3 THE THEOREM PROVERS

4 LEMMAS

APPENDIX

1 LINEAR LOGIC

1.1 Introduction

Linear Logic, introduced by Girard [6] is a refinement of classical and intuitionistic logic. Like intuitionistic logic it is constructive. It is however a more subtle logic giving tighter control of resources.

In linear logic the structural rules of contraction and weakening become logical rules. Contraction and weakening are still possible, but only on formulae preceded by a modal operator, ! or ?. Thus $!A \vdash A \otimes A$ is provable but $A \vdash A \otimes A$ is not. Likewise $A \parr A \vdash A$ is not provable while $A \parr A \vdash ?A$ is.

This change in structural rules leads naturally to two forms of conjunction and disjunction; one called **multiplicative** and the other **additive**. The additive forms require resource sharing, while the multiplicative forms disallow it.

Linear negation is introduced for economy, to transform two-sided sequents to one-sided sequents.

There is also a corresponding Intuitionistic Linear Logic with the restriction that only one formula may appear on the right.

Linear logic has number of applications in computer science:

Functional Programming:

Abramsky has shown that interpreting Linear Logic computationally throws new light on weak and eager evaluation, the multiplicative connectives corresponding to eager evaluation, the additive ones to lazy evaluation [1].

Girard and Lafont describe an abstract machine that has no need of garbage collection and has a natural lazy evaluation mechanism [7].

Parallel Computation:

Girard has shown that the connectives of linear logic have meanings in terms of parallel computation [7]. Nariso Mari-Oliet and Jose Meseguer have shown that there is a correspondence between Petri nets and linear logic theories [12].

Logic programming:

Hodas and Miller have shown that a fragment of linear logic can be used as a logic programming language. They describe a goal directed interpretation of that fragment and show that context splitting can be viewed as a process that takes a context, uses part of it, and returns the rest to be consumed elsewhere. We exploit this idea below. They also demonstrate applications in natural language parsing and databases [9].

1.2 Terminology

Capital letters will be used throughout to represent formulae. Lower case letters will represent atoms. The special symbols 1, 0, $\top$ and $\perp$ are also considered to be atomic. An underlined capital letter will represent a **context** or a multi-set of formulae. For example $\underline{A}$ represents a multi-set of formulae A1, ..., An.

A **queried formula** is a formula that has the operator ? as its main connective. A **queried context** is a context in which all formula are queried and is represented by a ? followed by an underlined capital letter. For example $?\underline{A}$ represents formulae ?A1, ..., ?An.

A **banged formula** is a formula that has the operator ! as its main connective. A **banged context** is a context in which all formula are banged and is represented by a ! followed by an underlined capital letter. For example $!\underline{A}$ represents formulae !A1, ..., !An.

In the following we use C, D ... as meta-variables for formulae, p, q ... for atoms.

1.3 The languages of CLL, MALL, ILL and IMALL

Classical linear Logic (CLL)

The syntax of a CLL formula is:

$$\begin{array}{ll}
\mathrm{CLL} ::= & \mathrm{CLL}\ \&\ \mathrm{CLL} \\
& \mid \mathrm{CLL} \parr \mathrm{CLL} \\
& \mid \mathrm{CLL} \otimes \mathrm{CLL} \\
& \mid \mathrm{CLL} \oplus \mathrm{CLL} \\
& \mid \mathrm{CLL} \multimap \mathrm{CLL} \\
& \mid\ !\mathrm{CLL} \\
& \mid\ ?\mathrm{CLL} \\
& \mid \mathrm{CLL}^{\perp} \\
& \mid a \mid b \mid \ldots \mid z \\
& \mid 1 \mid 0 \mid \top \mid \perp
\end{array}$$

Notational Equivalences:

$$\begin{array}{lcllcl}
C^{\perp\perp} & = & C & 0^{\perp} & = & \top \\
\perp^{\perp} & = & 1 & \top^{\perp} & = & 0 \\
1^{\perp} & = & \perp & (!C)^{\perp} & = & ?C^{\perp} \\
(?C)^{\perp} & = & !C^{\perp} & (C \otimes D)^{\perp} & = & C^{\perp} \parr D^{\perp} \\
(C \parr D)^{\perp} & = & C^{\perp} \otimes D^{\perp} & (C \,\&\, D)^{\perp} & = & C^{\perp} \oplus D^{\perp} \\
(C \oplus D)^{\perp} & = & C^{\perp} \,\&\, D^{\perp} & C \multimap D & = & C^{\perp} \parr D
\end{array}$$

There is no rule for linear negation. A negated non-atomic formula is simply a notation for an equivalent formula (in which linear negation is not the main connective). Likewise there is no rule for linear implication. $C \multimap D$ is considered to abbreviate $C^{\perp} \parr D$.

The syntax of a CLL sequent is $\vdash \underline{A}$, where $\underline{A}$ is a context consisting of CLL formulae.

(Since contexts are multi-sets, we may have several representations of a single sequent. These correspond to different permutations of elements).

Axioms of CLL:

$$\frac{}{\vdash p, p^{\perp}}\,\mathrm{I} \qquad\qquad \frac{}{\vdash 1}\,1 \qquad\qquad \frac{}{\vdash \top, \underline{A}}\,\top$$

Rules of CLL:

$$\frac{\vdash C, \underline{A1} \qquad \vdash D, \underline{A2}}{\vdash C \otimes D, \underline{A1}, \underline{A2}}\,\otimes \qquad\qquad \frac{\vdash C, \underline{A}}{\vdash C \oplus D, \underline{A}}\,1\oplus \qquad\qquad \frac{\vdash D, \underline{A}}{\vdash C \oplus D, \underline{A}}\,2\oplus$$

$$\frac{\vdash C, \underline{A} \qquad \vdash D, \underline{A}}{\vdash C \& D, \underline{A}}\,\& \qquad\qquad \frac{\vdash C, D, \underline{A}}{\vdash C \parr D, \underline{A}}\,\parr \qquad\qquad \frac{\vdash \underline{A}}{\vdash \bot, \underline{A}}\,\bot$$

$$\frac{\vdash ?\,C, ?\,C, \underline{A}}{\vdash ?\,C, \underline{A}}\,C? \qquad\qquad \frac{\vdash \underline{A}}{\vdash ?\,C, \underline{A}}\,W? \qquad\qquad \frac{\vdash C, \underline{A}}{\vdash ?\,C, \underline{A}}\,D?$$

$$\frac{\vdash C, ?\,\underline{A}}{\vdash !\,C, ?\,\underline{A}}\,! \qquad\qquad \frac{\vdash C, \underline{A1} \qquad \vdash C^{\perp}, \underline{A2}}{\vdash \underline{A1}, \underline{A2}}\,\mathrm{Cut}$$

Cut elimination holds for CLL: Any proof of a sequent in CLL using the Cut rule can be transformed into a proof without the cut rule [6].

Note that in CLL the context $\underline{A}$ representing formulae A1, A2, ..., An can be considered[1] to be a formula of the form $A1 \parr A2 \parr \ldots \parr An$.

[1] Lemmas 1.12 and 1.13

Multiplicative and Additive Linear Logic (MALL)

This is the fragment of Classical Linear Logic that excludes the modal operators ! and ? and their associated rules. Thus the syntax is simply a restricted form of the CLL syntax and the transformations and rules are a subset of those for CLL:

The syntax of a MALL formula is:

$$\begin{array}{ll} \mathrm{MALL} ::= & \mathrm{MALL} \;\&\; \mathrm{MALL} \\ & \mid \mathrm{MALL} \parr \mathrm{MALL} \\ & \mid \mathrm{MALL} \otimes \mathrm{MALL} \\ & \mid \mathrm{MALL} \oplus \mathrm{MALL} \\ & \mid \mathrm{MALL} \multimap \mathrm{MALL} \\ & \mid \mathrm{MALL}^{\perp} \\ & \mid a \mid b \mid \ldots \mid z \\ & \mid 1 \mid 0 \mid \top \mid \bot \end{array}$$

Notational Equivalences:

$C^{\perp\perp}$	=	C	$0^{\perp}$	=	$\top$
$\bot^{\perp}$	=	1	$\top^{\perp}$	=	0
$1^{\perp}$	=	$\bot$	$(C \otimes D)^{\perp}$	=	$C^{\perp} \parr D^{\perp}$
$(C \parr D)^{\perp}$	=	$C^{\perp} \otimes D^{\perp}$	$(C \,\&\, D)^{\perp}$	=	$C^{\perp} \oplus D^{\perp}$
$(C \oplus D)^{\perp}$	=	$C^{\perp} \,\&\, D^{\perp}$	$C \multimap D$	=	$C^{\perp} \parr D$

These are as before, omitting those involving the modal operators ! and ?.

There is no rule for linear negation. A negated non-atomic formula is simply a notation for an equivalent formula (in which linear negation is not the main connective). Likewise there is no rule for linear implication. $C \multimap D$ is considered to abbreviate $C^{\perp} \parr D$.

The syntax of a MALL sequent is $\vdash \underline{A}$, where $\underline{A}$ is a context consisting of MALL formulae. (Since contexts are multi-sets, we may have several representations of a single sequent. These correspond to different permutations of elements).

Axioms of MALL:

$$\frac{}{\vdash p, p^{\perp}}\; I \qquad\qquad \frac{}{\vdash 1}\; 1 \qquad\qquad \frac{}{\vdash \top, \underline{A}}\; \top$$

Rules of MALL:

$$\frac{\vdash C, \underline{A1} \qquad \vdash D, \underline{A2}}{\vdash C \otimes D, \underline{A1}, \underline{A2}}\; \otimes \qquad \frac{\vdash C, \underline{A}}{\vdash C \oplus D, \underline{A}}\; 1\oplus \qquad \frac{\vdash D, \underline{A}}{\vdash C \oplus D, \underline{A}}\; 2\oplus$$

$$\frac{\vdash C, \underline{A} \qquad \vdash D, \underline{A}}{\vdash C \,\&\, D, \underline{A}}\; \& \qquad \frac{\vdash C, D, \underline{A}}{\vdash C \parr D, \underline{A}}\; \parr \qquad \frac{\vdash \underline{A}}{\vdash \bot, \underline{A}}\; \bot$$

$$\frac{\vdash C, \underline{A1} \qquad \vdash C^{\perp}, \underline{A2}}{\vdash \underline{A1}, \underline{A2}}\; \mathrm{Cut}$$

Cut elimination holds for MALL [6].

Intuitionistic Linear Logic (ILL)

The syntax of an ILL Formula is

$$\begin{array}{ll} \text{ILL} ::= & \text{ILL} \,\&\, \text{ILL} \\ & \mid \text{ILL} \multimap \text{ILL} \\ & \mid \text{ILL} \otimes \text{ILL} \\ & \mid \text{ILL} \oplus \text{ILL} \\ & \mid\, !\,\text{ILL} \\ & \mid\, ?\,\text{ILL} \\ & \mid a \mid b \mid \ldots \mid z \\ & \mid 1 \mid 0 \mid \top \end{array}$$

The syntax of an ILL sequent is $\underline{A} \vdash G$ where G (the goal) is an ILL formula and $\underline{A}$ is a context consisting of ILL formulae.

(Since contexts are multi-sets, we may have several representations of a single sequent. These correspond to different permutations of elements).

Axioms of ILL:

$$\frac{}{p \vdash p}\,I \qquad \frac{}{\vdash 1}\,1$$

$$\frac{}{\underline{A}, 0 \vdash G}\,0 \qquad \frac{}{\underline{A} \vdash \top}\,\top$$

Rules of ILL:

$$\frac{\underline{A} \vdash G}{\underline{A}, 1 \vdash G}\,1L \qquad \frac{\underline{A1} \vdash C \quad D, \underline{A2} \vdash G}{C \multimap D, \underline{A1}, \underline{A2} \vdash G}\,{\multimap}L \qquad \frac{\underline{A}, C \vdash D}{\underline{A} \vdash C \multimap D}\,{\multimap}R$$

$$\frac{\underline{A}, C \vdash G \quad \underline{A}, D \vdash G}{\underline{A}, C \oplus D \vdash G}\,{\oplus}L \qquad \frac{\underline{A} \vdash C}{\underline{A} \vdash C \oplus D}\,{\oplus}R1 \qquad \frac{\underline{A} \vdash D}{\underline{A} \vdash C \oplus D}\,{\oplus}R2$$

$$\frac{\underline{A}, C \vdash G}{\underline{A}, C \& D \vdash G}\,\&L1 \qquad \frac{\underline{A}, D \vdash G}{\underline{A}, C \& D \vdash G}\,\&L2 \qquad \frac{\underline{A} \vdash C \quad \underline{A} \vdash D}{\underline{A} \vdash C \& D}\,\&R$$

$$\frac{\underline{A}, C, D \vdash G}{\underline{A}, C \otimes D \vdash G}\,{\otimes}L \qquad \frac{\underline{A1} \vdash C \quad \underline{A2} \vdash D}{\underline{A1}, \underline{A2} \vdash C \otimes D}\,{\otimes}R \qquad \frac{\underline{A}, !C, !C \vdash G}{\underline{A}, !C \vdash G}\,C!$$

$$\frac{\underline{A} \vdash G}{\underline{A}, !C \vdash G}\,W! \qquad \frac{\underline{A}, C \vdash G}{\underline{A}, !C \vdash G}\,D! \qquad \frac{!\underline{A} \vdash G}{!\underline{A} \vdash !G}\,!R$$

$$\frac{\underline{A1} \vdash C \quad \underline{A2}, C \vdash G}{\underline{A1}, \underline{A2} \vdash G}\,Cut$$

Cut elimination holds for ILL: Any proof of a sequent in ILL using the Cut rule can be transformed into a proof without the cut rule [6].

In ILL the context $\underline{A}$ representing formulae A1, A2, ..., An can be considered[2] to be a formula of the form $A1 \otimes A2 \otimes \ldots \otimes An$.

[2] Lemmas 2.17 and 2.18

Intuitionistic Multiplicative and Additive Linear Logic (IMALL)

This is the fragment of Intuitionistic Linear Logic that excludes the modal operators ! and ? and their associated rules. Thus the syntax is simply a restricted form of the ILL syntax and the transformations and rules are a subset of those for ILL:

The syntax of an IMALL Formula is:

$$
\begin{array}{ll}
\text{IMALL} ::= & \text{IMALL} \;\&\; \text{IMALL} \\
 & \mid \text{IMALL} \multimap \text{IMALL} \\
 & \mid \text{IMALL} \otimes \text{IMALL} \\
 & \mid \text{IMALL} \oplus \text{IMALL} \\
 & \mid a \mid b \mid \ldots \mid z \\
 & \mid 1 \mid 0 \mid \top
\end{array}
$$

The syntax of an IMALL sequent is $\underline{A} \vdash G$ where G (the goal) is an IMALL formula and $\underline{A}$ is a context consisting of IMALL formulae.

(Since contexts are multi-sets, we may have several representations of a single sequent. These correspond to different permutations of elements).

Axioms of IMALL:

$$\frac{}{p \vdash p}\ I \qquad \frac{}{\vdash 1}\ 1$$

$$\frac{}{\underline{A}, 0 \vdash G}\ 0 \qquad \frac{}{\underline{A} \vdash \top}\ \top$$

Rules of IMALL:

$$\frac{\underline{A} \vdash G}{\underline{A}, 1 \vdash G}\ 1L \qquad \frac{\underline{A1} \vdash C \quad D, \underline{A2} \vdash G}{C \multimap D, \underline{A1}, \underline{A2} \vdash G}\ \multimap L \qquad \frac{\underline{A}, C \vdash D}{\underline{A} \vdash C \multimap D}\ \multimap R$$

$$\frac{\underline{A}, C \vdash G \quad \underline{A}, D \vdash G}{\underline{A}, C \oplus D \vdash G}\ \oplus L \qquad \frac{\underline{A} \vdash C}{\underline{A} \vdash C \oplus D}\ \oplus R1 \qquad \frac{\underline{A} \vdash D}{\underline{A} \vdash C \oplus D}\ \oplus R2$$

$$\frac{\underline{A}, C \vdash G}{\underline{A}, C \& D \vdash G}\ \&L1 \qquad \frac{\underline{A}, D \vdash G}{\underline{A}, C \& D \vdash G}\ \&L2 \qquad \frac{\underline{A} \vdash C \quad \underline{A} \vdash D}{\underline{A} \vdash C \& D}\ \&R$$

$$\frac{\underline{A}, C, D \vdash G}{\underline{A}, C \otimes D \vdash G}\ \otimes L \qquad \frac{\underline{A1} \vdash C \quad \underline{A2} \vdash D}{\underline{A1}, \underline{A2} \vdash C \otimes D}\ \otimes R$$

$$\frac{\underline{A}1 \vdash C \quad \underline{A}2, C \vdash G}{\underline{A}1, \underline{A}2 \vdash G}\ \text{Cut}$$

Cut elimination holds for IMALL [6].

1.4 Linearised Intuitionistic Logics

IL embedded in CLL (ILinCLL)

Intuitionistic logic can be embedded into CLL [8]. This embedding works at the level of proofs and sequents. Thus it is possible to translate an IL formula to a CLL formula and to translate an IL problem into a CLL problem. The recursively defined translation[3] is given below:

$(C \vee D)^*$	=	$!\,C^* \oplus\, !\,D^*$	$(C \,\&\, D)^*$	=	$C^* \,\&\, D^*$
$(C \rightarrow D)^*$	=	$?\,C^{*\perp} \parr D^*$	$({\sim}\, C)^*$	=	$?\,C^{*\perp} \parr 0$
$\wedge^*$	=	0	p^*	=	p

$$(A1, \ldots, An \vdash G)^* = \vdash ?(\,(A1^*)^\perp), \ldots, ?(\,(An^*)^\perp), G^*$$

Rather than implementing a translator that initially turns a intuitionistic problem into a CLL one, it was decided that the translation should take place only as required (i.e to ensure that the main connective of a formula is always a linear one, so that a CLL rule may be applied). The reason for doing this is to be able to see more clearly the relationship between IL and CLL. Also a reasonably sized (in terms of printed length on the screen) IL problem will translate to a large CLL problem

This resulted in a hybrid logic which I have called ILinCLL. ILinCLL is identical to CLL except that linear formulae may have intuitionistic subformulae. These intuitionistic subformulae are simply a meta-notation for the equivalent CLL formulae, and are marked as such by the brackets '⟦' and '⟧'.

The syntax of ILinCLL Formula is:

$$\begin{array}{lll}
\text{ILinCLL} ::= & \text{ILinCLL} \,\&\, \text{ILinCLL} & \\
& \mid \text{ILinCLL} \parr \text{ILinCLL} & \\
& \mid \text{ILinCLL} \otimes \text{ILinCLL} & \\
& \mid \text{ILinCLL} \oplus \text{ILinCLL} & \\
& \mid \text{ILinCLL} \multimap \text{ILinCLL} & \\
& \mid\, !\,\text{ILinCLL} \mid ?\,\text{ILinCLL} & \\
& \mid \text{ILinCLL}^{\perp} & \\
& \mid a \mid b \mid \ldots \mid z & \\
& \mid 1 \mid 0 \mid \top \mid \perp & \\
& \mid [\![\text{IL}]\!] \,\&\, [\![\text{IL}]\!] & \text{IL} ::= \ \text{IL} \,\&\, \text{IL} \\
& \mid [\![\text{IL}]\!] \parr [\![\text{IL}]\!] & \qquad\ \mid \text{IL} \vee \text{IL} \\
& \mid [\![\text{IL}]\!] \otimes [\![\text{IL}]\!] & \qquad\ \mid \text{IL} \rightarrow \text{IL} \\
& \mid [\![\text{IL}]\!] \oplus [\![\text{IL}]\!] & \qquad\ \mid \text{IL} \rightarrow \text{IL} \\
& \mid [\![\text{IL}]\!] \multimap [\![\text{IL}]\!] & \qquad\ \mid a \mid b \mid \ldots \mid z \mid \wedge \\
& \mid\, !\,[\![\text{IL}]\!] \mid ?\,[\![\text{IL}]\!] & \\
& \mid [\![\text{IL}]\!]^{\perp} &
\end{array}$$

Notational Equivalences:

$C^{\perp\perp}$	=	C	$(C \otimes D)^{\perp}$	=	$C^{\perp} \parr D^{\perp}$
$(C \parr D)^{\perp}$	=	$C^{\perp} \otimes D^{\perp}$	$(C \,\&\, D)^{\perp}$	=	$C^{\perp} \oplus D^{\perp}$
$(C \oplus D)^{\perp}$	=	$C^{\perp} \,\&\, D^{\perp}$	$(\,!\,C\,)^{\perp}$	=	$?\,C^{\perp}$
$(\,?\,C\,)^{\perp}$	=	$!\,C^{\perp}$	$1^{\perp}$	=	$\perp$
$\perp^{\perp}$	=	1	$\top^{\perp}$	=	0
$0^{\perp}$	=	$\top$	$C \multimap D$	=	$C^{\perp} \parr D$
$[\![C \,\&\, D]\!]$	=	$[\![C]\!] \,\&\, [\![D]\!]$	$[\![C \rightarrow D]\!]$	=	$?\,[\![C]\!]^{\perp} \parr [\![D]\!]$
$[\![{\sim}\, C]\!]$	=	$?\,[\![C]\!]^{\perp} \parr 0$	$[\![\wedge]\!]$	=	0
$[\![C \vee D]\!]$	=	$!\,[\![C]\!] \oplus\, !\,[\![D]\!]$	$[\![p]\!]$	=	p

[3] $(C \rightarrow D)^* = !C^* \multimap D^*$ and $({\sim}C)^* = !C^* \multimap 0$

The rules and axioms are the same as those of CLL. As can be seen, the equivalences are a superset of the CLL ones. Likewise the syntax of ILinCLL formulae is an extension of that of CLL formulae.

The syntax of a ILinCLL sequent is $\vdash \underline{A}$, where $\underline{A}$ is a context consisting of ILinCLL formulae.

(Since contexts are multi-sets, we may have several representations of a single sequent. These correspond to different permutations of elements).

<u>IL embedded in ILL (ILinILL)</u>

Intuitionistic logic can also be embedded in a similar way into ILL [8]:

$$\begin{array}{lcllcl}
(C \vee D)^* & = & !\,C^* \oplus\, !\,D^* & (C \,\&\, D)^* & = & C^* \,\&\, D^* \\
(C \rightarrow D)^* & = & ?\,C^{*\perp} \parr D^* & \sim C & = & ?\,C^{*\perp} \parr 0 \\
\curlywedge^* & = & 0 & p^* & = & p
\end{array}$$

$$(A1, \ldots\ldots, An \vdash G)^* = !\,(A1^*), \ldots\ldots, !(An^*) \vdash G^*$$

The syntax of ILinILL Formula is

$$\begin{array}{lll}
\text{ILinILL} & ::= & \text{ILinILL} \,\&\, \text{ILinILL} \\
 & & \mid \text{ILinILL} \multimap \text{ILinILL} \\
 & & \mid \text{ILinILL} \otimes \text{ILinILL} \\
 & & \mid \text{ILinILL} \oplus \text{ILinILL} \\
 & & \mid \text{ILinILL} \,\&\, \text{ILinILL} \\
 & & \mid\, !\,\text{ILinILL} \\
 & & \mid\, ?\,\text{ILinILL} \\
 & & \mid \text{ILinILL}^{\perp} \\
 & & \mid a \mid b \ldots\ldots y \mid z \\
 & & \mid 1 \mid 0 \mid \top \mid \perp \\
 & & \mid \llbracket \text{IL} \rrbracket \,\&\, \llbracket \text{IL} \rrbracket \\
 & & \mid \llbracket \text{IL} \rrbracket \multimap \llbracket \text{IL} \rrbracket \\
 & & \mid \llbracket \text{IL} \rrbracket \otimes \llbracket \text{IL} \rrbracket \\
 & & \mid \llbracket \text{IL} \rrbracket \oplus \llbracket \text{IL} \rrbracket \\
 & & \mid\, !\,\llbracket \text{IL} \rrbracket \\
 & & \mid\, ?\,\llbracket \text{IL} \rrbracket \\
 & & \mid \llbracket \text{IL} \rrbracket^{\perp}
\end{array}$$

$$\begin{array}{lll}
\text{IL} & ::= & \text{IL} \,\&\, \text{IL} \\
 & & \mid \text{IL} \vee \text{IL} \\
 & & \mid \text{IL} \rightarrow \text{IL} \\
 & & \mid \sim \text{IL} \\
 & & \mid a \mid b \mid \ldots \mid z \mid \curlywedge
\end{array}$$

Notational Equivalences:

$$\begin{array}{lcllcl}
\llbracket C \vee D \rrbracket & = & !\,C \oplus\, !\,D & \llbracket C \vee D \rrbracket & = & !\,\llbracket C \rrbracket \oplus\, !\,\llbracket D \rrbracket \\
\llbracket C \,\&\, D \rrbracket & = & \llbracket C \rrbracket \,\&\, \llbracket D \rrbracket & \llbracket C \rightarrow D \rrbracket & = & !\,\llbracket C \rrbracket \multimap \llbracket D \rrbracket \\
\llbracket \sim C \rrbracket & = & !\,\llbracket C \rrbracket \multimap 0 & \llbracket \curlywedge \rrbracket & = & 0 \\
\llbracket p \rrbracket & = & p & & &
\end{array}$$

The rules and axioms are the same as those of ILL. The syntax of ILinILL formulae is an extension of that of ILL formulae.

The syntax of an ILinILL sequent is $\underline{A} \vdash G$ where G (the goal) is an ILinILL formula and $\underline{A}$ consists of ILinILL formulae.

(Since contexts are multi-sets, we may have several representations of a single sequent. These correspond to different permutations of elements).

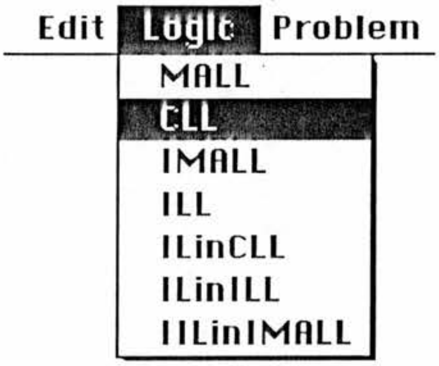
Lincoln et al have shown that it is possible to embed a variant of Implicative Intuitionistic Logic into IMALL [10]. This embedding is also implemented and described later.

2 THE PROOF EDITOR

2.1 A Demonstration

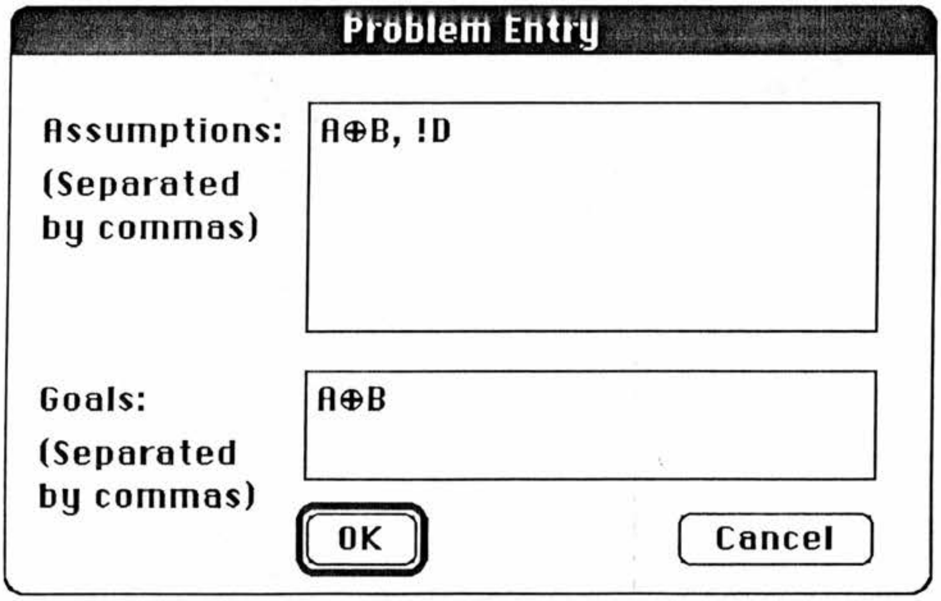
In this section I shall run through a demonstration which aims to show the major features of the Proof Editor.

The application is started in the normal manner for a Macintosh application, by double clicking on the program icon. We are then presented with a window entitled 'Proof'. The first thing to do is to decide which Logic we wish to work in; the choice is between CLL, MALL, ILL, IMALL, ILinCLL, ILinILL and IILinIMALL. The choice is made by selecting the logic from the Logic menu:



During this demonstration we shall assume that CLL has been selected.

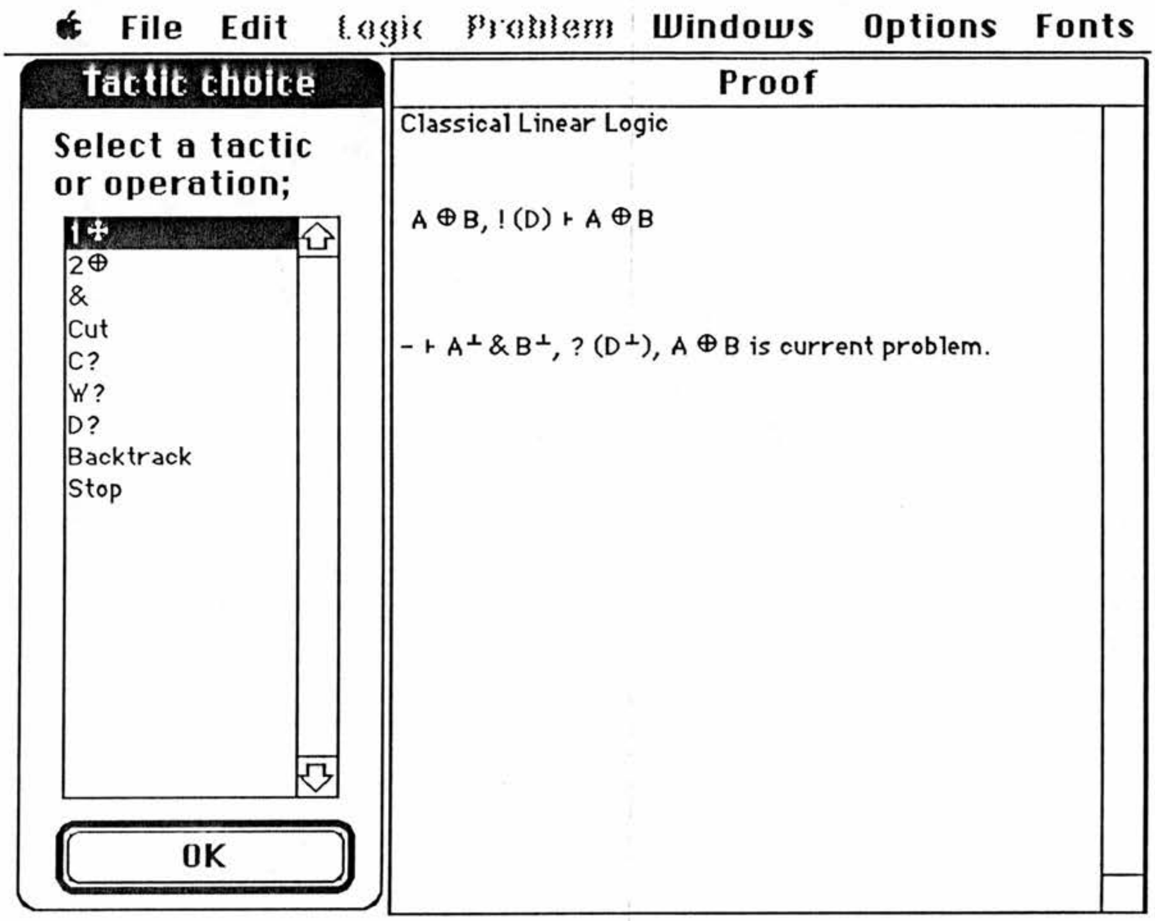
The initial conjecture can be entered in two ways, either by selecting some text, representing a sequent, and selecting 'Front Window' from the Problem menu, or by selecting 'Dialog...' from the 'Problem' menu:


Problem Entry

Assumptions: (Separated by commas) A⊕B, !D

Goals: (Separated by commas) A⊕B

The formulae in the assumptions field are those formulae which appear on the left of a two sided sequent. Since we are working in CLL these will be linearly negated and brought to the right hand side. If we press OK, the initial conjecture is written into the Proof window and the 'Tactic choice' dialog appears:



Classical Linear Logic

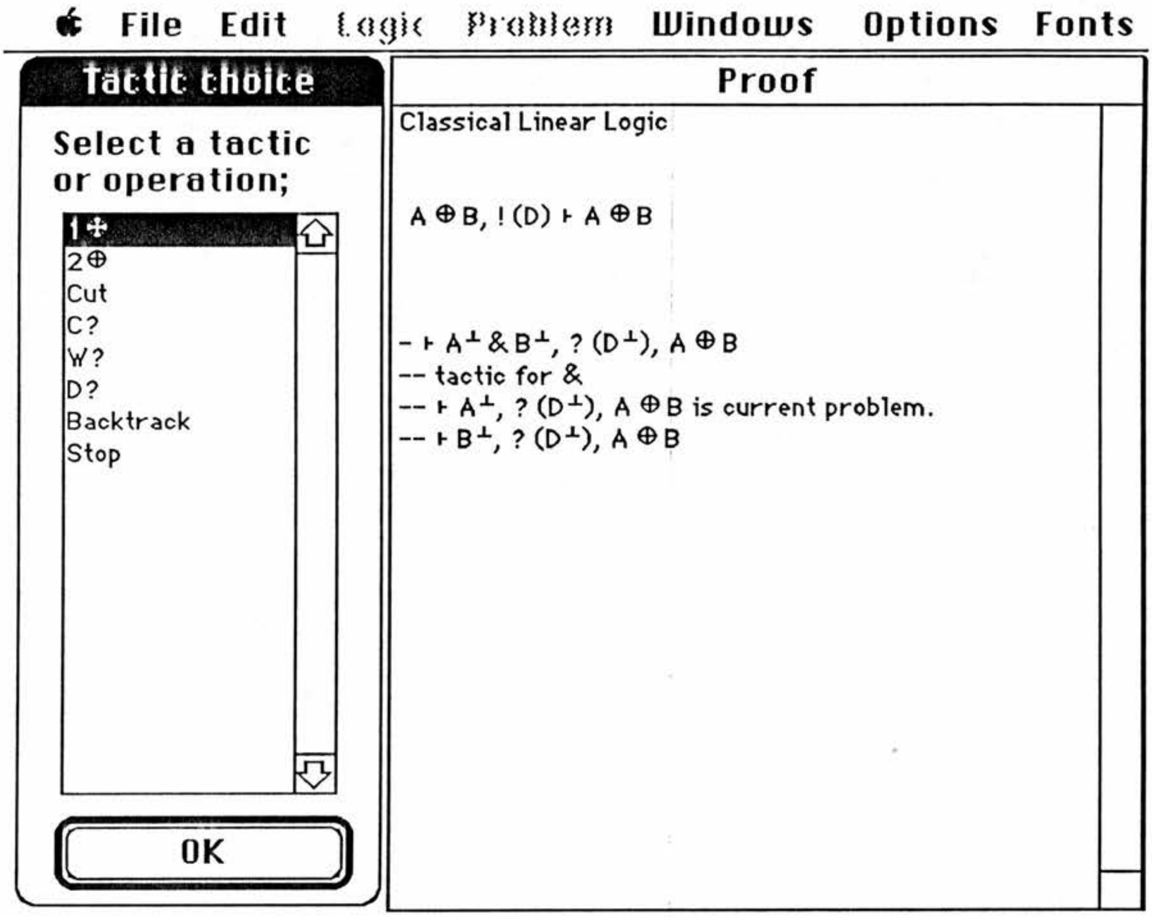
A ⊕ B, ! (D) ⊢ A ⊕ B

- ⊢ $A^{\perp}$ & $B^{\perp}$, ? ($D^{\perp}$), A ⊕ B is current problem.

The formula A ⊕ B has first been linearly negated as it appeared on the left of the sequent. The formula $(A \oplus B)^{\perp}$ was then transformed into $A^{\perp}$ & $B^{\perp}$. This is necessary as there is no rule for the $^{\perp}$ connective.

The 'Tactics' dialog contains all the rules of CLL. If 'Applicable Tactics' is selected from the Options menu only those tactics applicable to the current problem will be displayed.

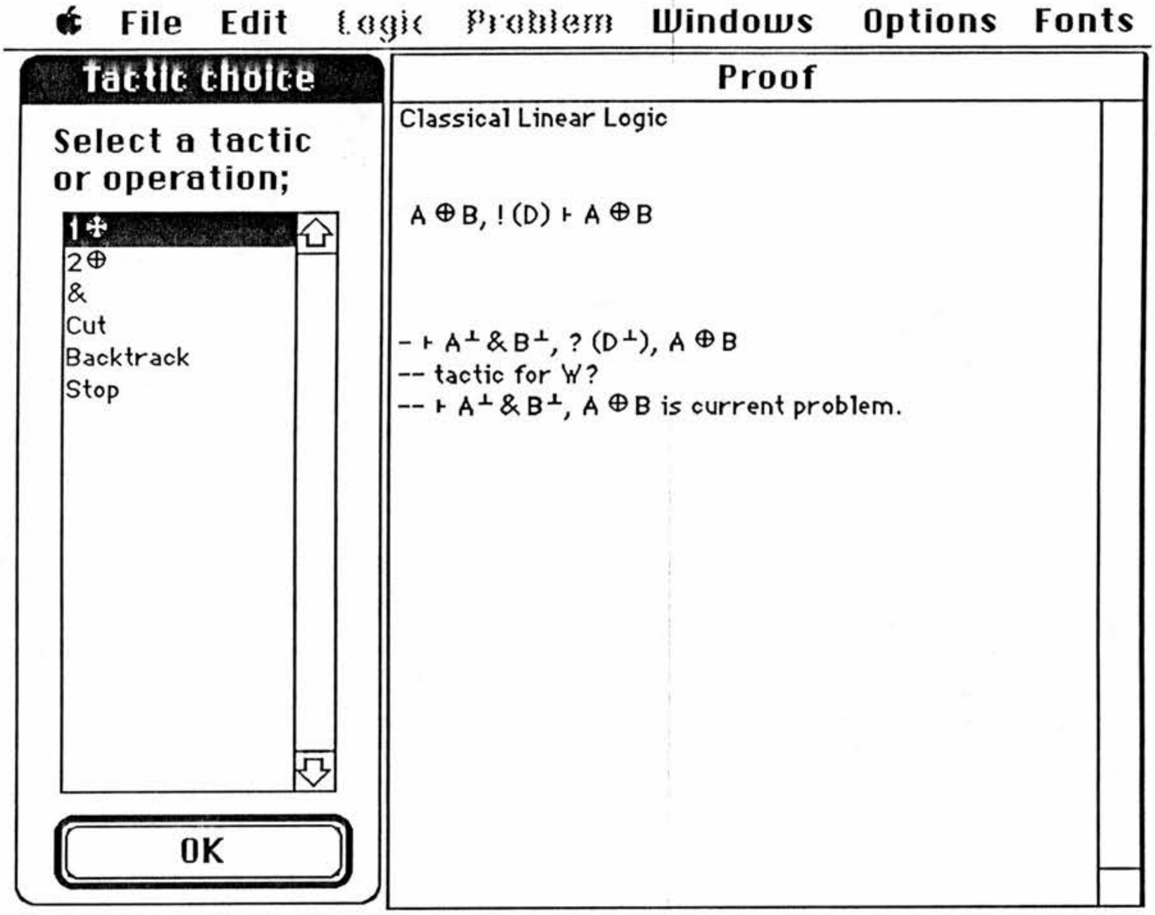
We are now ready to start the proof. This is done by successively applying tactics to a problem, to produce subproblems. The proof is completed when all the subproblems are trivial. The tactics correspond directly to the rules of CLL. To apply the tactic for & to the problem simply select & in the scrolling menu and click on the OK button. Two new subproblems created by this tactic are written to the proof window:



```
Classical Linear Logic


 A ⊕ B, ! (D) ⊢ A ⊕ B



- ⊢ A⊥ & B⊥, ? (D⊥), A ⊕ B
-- tactic for &
-- ⊢ A⊥, ? (D⊥), A ⊕ B is current problem.
-- ⊢ B⊥, ? (D⊥), A ⊕ B
```

The application of any tactic can be undone by selecting the operation 'Backtrack'. When we do this the newly created subproblems are erased from the screen. The user has full control over the construction of the proof and all operations are 'undoable'.

Although we could continue this proof, it is better to backtrack and apply W?, so that this tactic need be applied only once. Applying the tactic for W? instead, gives:

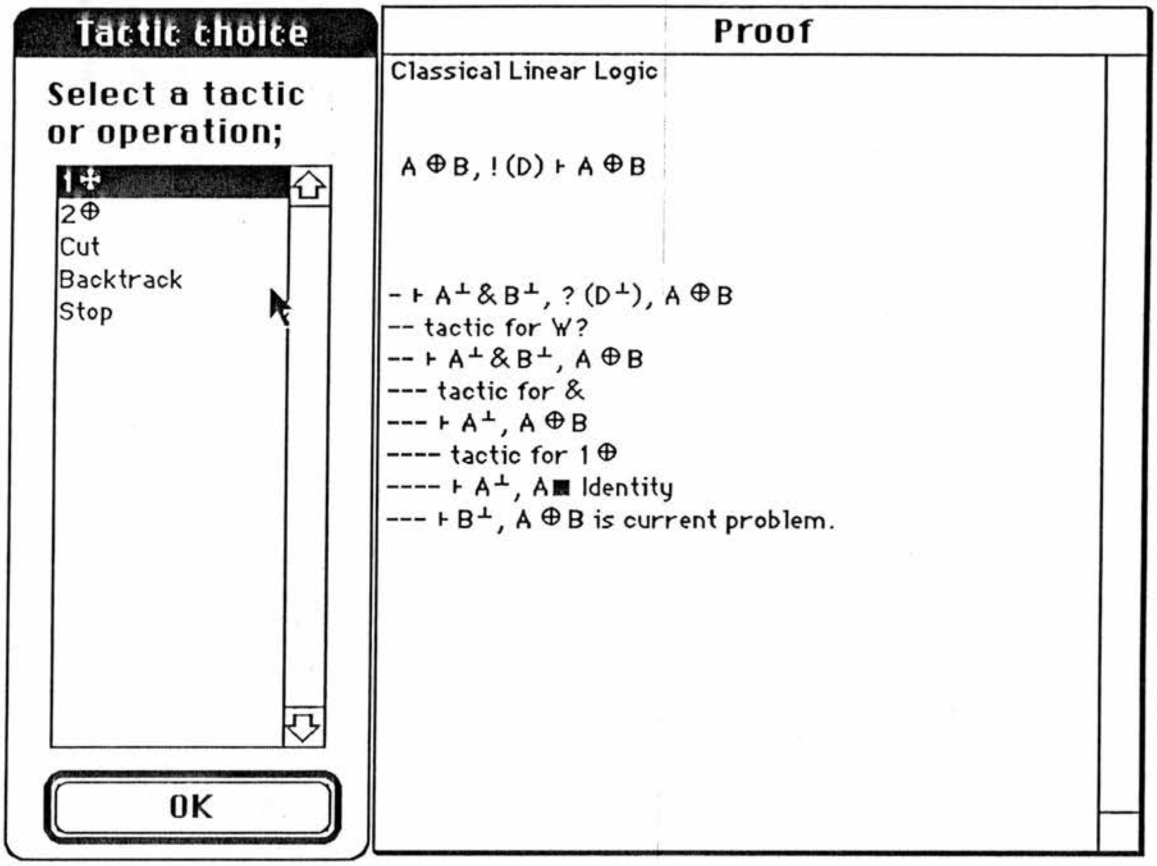


```
Classical Linear Logic

 A ⊕ B, ! (D) ⊢ A ⊕ B

- ⊢ A⊥ & B⊥, ? (D⊥), A ⊕ B
-- tactic for W?
-- ⊢ A⊥ & B⊥, A ⊕ B is current problem.
```

We now apply the tactic for & creating two new subproblems. Applying the tactic for 1⊕ solves the first subproblem which has '■' written after it.

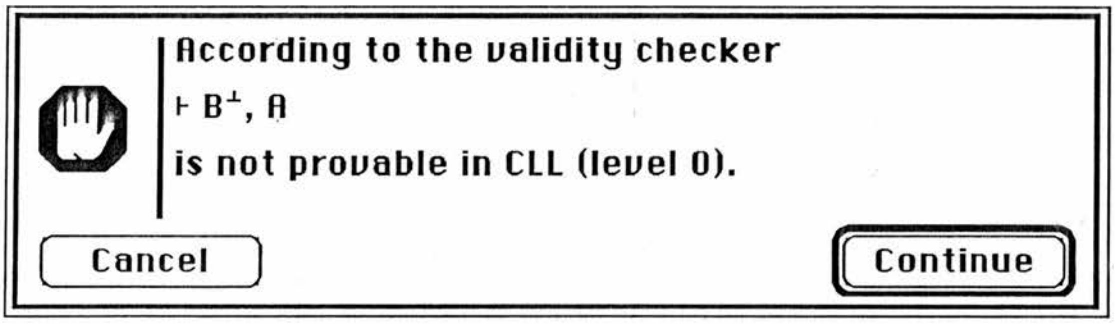
If one selects 'Axioms as Tactics' from the 'Options' menu then trivial problems are not solved automatically but must have the tactic for the appropriate Axiom applied to them. Thus T ⊢ T can be solved by the tactic for T or the tactic for I.

We are now working on the second subproblem:



```
Classical Linear Logic


 A ⊕ B, ! (D) ⊢ A ⊕ B



- ⊢ A⊥ & B⊥, ? (D⊥), A ⊕ B
-- tactic for ⅋?
-- ⊢ A⊥ & B⊥, A ⊕ B
--- tactic for &
--- ⊢ A⊥, A ⊕ B
---- tactic for 1 ⊕
---- ⊢ A⊥, A■ Identity
--- ⊢ B⊥, A ⊕ B is current problem.
```

If we try applying 1⊕ to the current problem the following dialog appears:

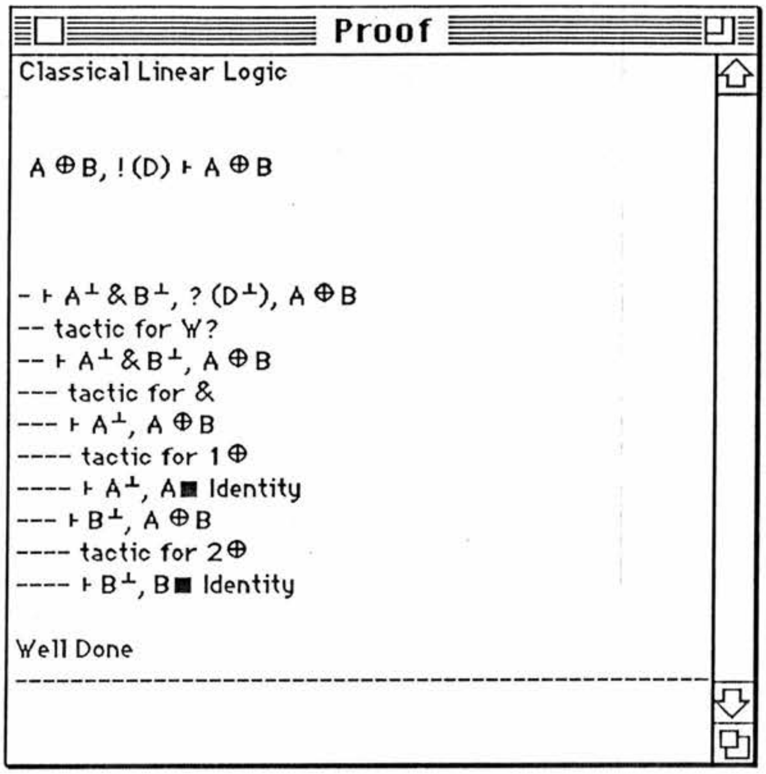


According to the validity checker

⊢ B⊥, A

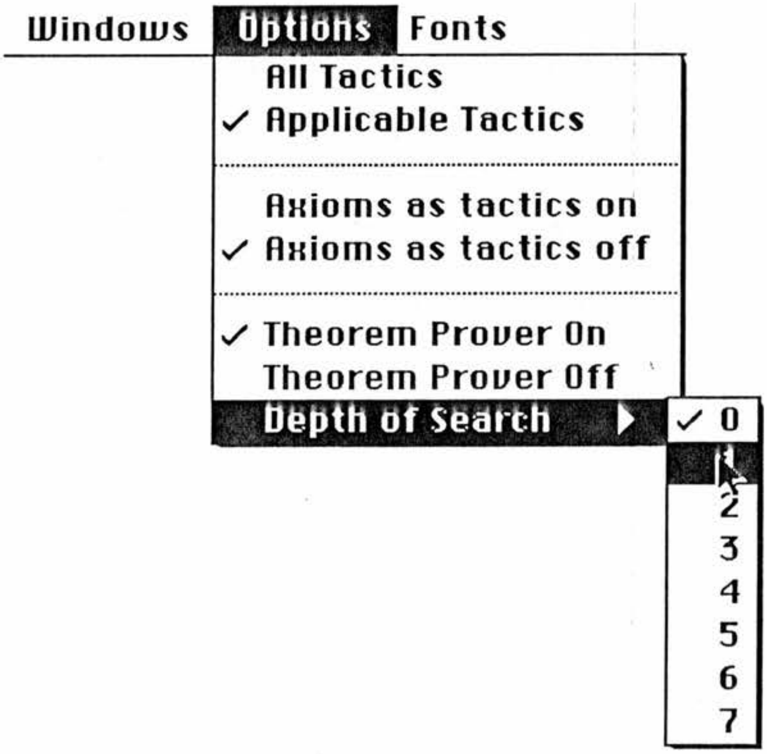
is not provable in CLL (level 0).

We click Cancel and apply 2⊕ instead. The initial problem is now solved:



```
Classical Linear Logic


 A ⊕ B, ! (D) ⊢ A ⊕ B



- ⊢ A⊥ & B⊥, ? (D⊥), A ⊕ B
-- tactic for W?
-- ⊢ A⊥ & B⊥, A ⊕ B
--- tactic for &
--- ⊢ A⊥, A ⊕ B
---- tactic for 1 ⊕
---- ⊢ A⊥, A■ Identity
--- ⊢ B⊥, A ⊕ B
---- tactic for 2⊕
---- ⊢ B⊥, B■ Identity

Well Done
---------------------------------------------
```

The theorem prover will warn of any attempt to solve unsolvable problems. If the theorem prover is sure that the problem is unsolvable then it will inform you that the problem 'is' unsolvable. If working in an undecidable logic and the problem contains modal operators you will be informed that the problem 'may be' unsolvable. In either case you may decide to carry on with the proof. The level of certainty expressed by the theorem prover will depend on the search depth of the theorem prover. This can be set in the Options Menu. The higher this figure, the longer the theorem prover will take to terminate on each occasion. The theorem prover can of course be turned off.



2.2 Implementation issues

The programming language Prolog was chosen as the Language for this project.

The reason for this is that the in-built unification algorithm allows easy matching of formulae against rules, and one has the ability to backtrack without having to implement a stack. This is important as the user must be able to undo previous operations up to an arbitrary depth.

Also, being declarative, a Prolog program is shorter and easier to write than a similar imperative program. Definite Clause Grammars allow one to write parsers and unparsers with great ease. The loss of efficiency is of little concern as the time spent on computation will be less than the time spent on user interface.

User defined operators and the ease with which meta-interpreters can be written allows code to be presented in a very natural way. The syntax allows easy representation of lists and trees, which means easy representation of sequents and proofs.

It is often easy to check that a Prolog Program fulfills its specification. This is particularly important when implementing theorem provers.

Another reason is that the use of Prolog had proved successful in the implementation of MacLogic, and parts of that program could be used for this project [4].

However, the unusual requirements for a proof editor make some essentially procedural parts of the program, concerned with user interface, awkward to code in Prolog. These however form only a small part of the program. Also for a relatively large project such as this, Prolog's lack of scoping is a disadvantage as it makes debugging and data abstraction difficult.

MacProlog™ was chosen as it gives easy access to the user friendly aspects of the Macintosh™ computer. Consistency is maintained with other Macintosh applications, and the guide-lines given by Apple [2]. All interactions with the proof editor take place by way of menus and dialog boxes.

2.3 Data Structures and the Parser/Unparser

A CLL problem has the following structure: ⊢ A1, ..., An where A1, ..., An are formulae. This is represented in the proof editor by the Prolog data structure: ⊢ [A1, ..., An].

An ILL problem has the form A1, ..., An ⊢ G where A1, ..., An, G are formulae. This is represented by the Prolog data structure: [A1, ..., An] ⊢ G.

(In the CLL the turnstile is included to differentiate CLL and ILL problems. ⊢ _V will not unify with an ILL problem whereas _V will. This helps to overcome the lack of typing in standard Prolog.)

Formulas are Prolog terms.

The code for parsing and unparsing formulae is the same as that used in MacLogic with additional code for the linear connectives. The precedence and associativity of the linear connectives is given below. These follow the convention that ⊸ associates to the right and ! binds more strongly than ⊸:

```
prefix(   !,    750,    fy      ).
prefix(   ?,    750,    fy      ).
postfix(  ⊥,    750,    fy      ).

infix(  ⅋,     950,    yfx     ).
infix(  ⊗,     950,    yfx     ).
infix(  ⊕,     950,    yfx     ).
infix(  ⊸,     1050,   xfy     ).
infix(  &,     950,    yfx     ).
```

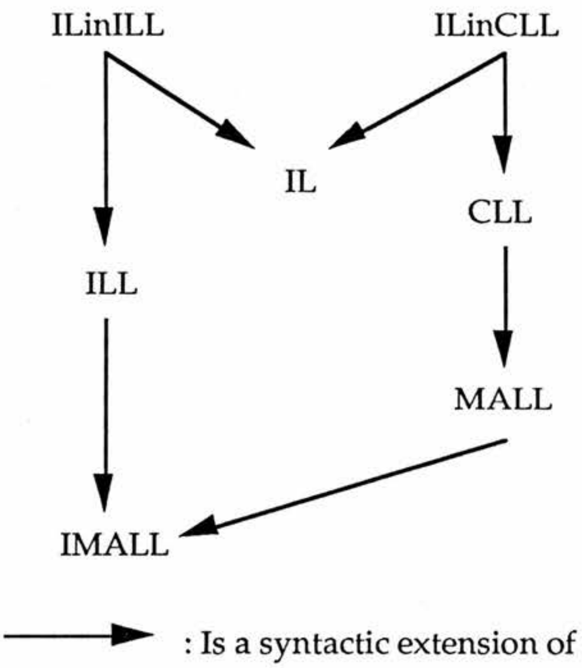
The precedence and associativity of the intuitionistic connectives is given below:

```
prefix(¬,        750,   fy      ).

infix(  ∨,       1000,  yfx     ).
infix(  &,       950,   yfx     ).
infix(  →,       1050,  xfy     ).
```

The parser will attempt to parse a string containing any of these connectives. Thus, to ensure that the string represents a valid formula for any particular logic, it is first checked for any unacceptable characters.

```
accept( _, [], [] ).
accept(Logic, [ C | T1 ], [ C | T2] ) :-
     symbol_for([C], Logic ), !,
     accept(Logic, T1, T2).
accept( Logic,  [C | T1], T2) :-
     ignorable(C), !,
     accept(Logic, T1, T2).
```

The following diagram shows the syntactic relationships between the various logics:



```
ignorable( 32 ).                % Space
ignorable( 202 ).               % Option space
ignorable( 9).                  % Tab
ignorable( 227).                % Shift Option W, i.e. tiny space
ignorable( 13).                 % Return

symbol_for([A], _) :-
     A ≥ 65,
     A ≤ 90.
symbol_for("(", _).
symbol_for(")", _).
symbol_for("[", _).
symbol_for("]", _).     % Belong to all the logics
symbol_for("⊗", 'IMALL').
symbol_for("⊕", 'IMALL').
symbol_for("&", 'IMALL').
symbol_for("⊸", 'IMALL').
symbol_for("1", 'IMALL').
symbol_for("0", 'IMALL').
symbol_for("T", 'IMALL').

'ILL' is_a_syntactic_extension_of 'IMALL'.
symbol_for("!", 'ILL').

'ILinILL' is_a_syntactic_extension_of 'ILL'
'ILinILL' is_a_syntactic_extension_of 'IL'.
symbol_for("⟦", 'ILinILL').
symbol_for("⟧", 'ILinILL').

symbol_for("&", 'IL').
symbol_for("→", 'IL').
symbol_for("∧", 'IL').
symbol_for("∨", 'IL').
symbol_for("~", 'IL').

symbol_for("⅋", 'MALL').
'MALL' is_a_syntactic_extension_of 'IMALL'.

'CLL' is_a_syntactic_extension_of 'MALL'.
symbol_for("!", 'CLL').
symbol_for("?", 'CLL').

'ILinCLL' is_a_syntactic_extension_of 'CLL'.
'ILinCLL' is_a_syntactic_extension_of 'IL'.
symbol_for("⟦", 'ILinCLL').
symbol_for("⟧", 'ILinCLL').

symbol_for(X, Logic1) :-
      Logic1 is_a_syntactic_extension_of Logic,
      symbol_for(X, Logic).
```

The parser needs to be informed of the sequent type of the various logics as they are entered. For instance a CLL problem is entered as a two sided sequent with many formulae on both the left and the right. This later becomes transformed into a one sided sequent. An ILL problem is entered with just one formula on the right:

```
initial_sequent_type('CLL', list ⊢ list).
initial_sequent_type('ILL', list ⊢ formula).
initial_sequent_type('MALL', list ⊢ list).
initial_sequent_type('IMALL', list ⊢ formula).
initial_sequent_type('ILinCLL', list ⊢ formula).
initial_sequent_type('ILinILL', list ⊢ formula).
initial_sequent_type('IL', list ⊢ formula).
```

ILinCLL and ILinILL sequents may only have IL formulae when initially entered:

```
initial_syntax('ILinCLL', 'IL').
initial_syntax('ILinILL', 'IL').
```

The unparser is also similar to that in MacLogic. It is designed to omit brackets that are unnecessary and would make formulae appear confusing. Thus for example the formula $(?(A^{\perp})\multimap(B^{\perp}))$ unparses as $?\,A^{\perp}\multimap B^{\perp}$.

2.4 Derivation of Tactics and implementation

From the rules for CLL, MALL, ILL and MALL we need to derive 'tactics' which can be applied to problems to form new problems. For a formal description of Tactics see [3].

The Tactics of MALL:

$(\underline{A} / C)$ denotes the context $\underline{A}$ with one occurrence of C removed.
$C :: \underline{A}$ denotes the context containing C and all the formulae in the context $\underline{A}$.

Tactic for 'Cut':
$\vdash A$ if $\vdash C, \underline{A1}$ and $\vdash C^{\perp}, \underline{A2}$ (where $\underline{A1}, \underline{A2}$ is a partition of $\underline{A}$ and C is any formula).

Tactic for &:
$\vdash \underline{A}$ if $C \& D \in \underline{A}$ and $\vdash C :: (\underline{A} / C\&D)$ and $\vdash D :: (\underline{A} / C\&D)$.

Tactic for $1\oplus$:
$\vdash \underline{A}$ if $C \oplus D \in \underline{A}$ and $\vdash C :: (\underline{A} / C \oplus D)$.

Tactic for $2\oplus$:
$\vdash \underline{A}$ if $C \oplus D \in \underline{A}$, and $\vdash D :: (\underline{A} / C \oplus D)$.

Tactic for $\otimes$:
$\vdash \underline{A}$ if $C \otimes D \in \underline{A}$ and $\vdash C, \underline{A1}$ and $\vdash D, \underline{A2}$ (where $\underline{A1}, \underline{A2}$ is a partition of $(\underline{A} / C \otimes D)$).

Tactic for ⅋ :
$\vdash \underline{A}$ if C ⅋ D $\in \underline{A}$ and $\vdash C :: D :: (\underline{A} /$ C ⅋ D $)$.

Tactic for Perp :
$\vdash \underline{A}$ if $\perp \in \underline{A}$ and $\vdash (\underline{A} / \perp)$.

The tactics of CLL:

Tactic for D? :
$\vdash \underline{A}$ if $?C \in \underline{A}$ and $\vdash C :: (\underline{A} / ?C)$.

Tactic for W? :
$\vdash \underline{A}$ if $?C \in \underline{A}$ and $\vdash (\underline{A} / ?C)$.

Tactic for C? :
$\vdash \underline{A}$ if $?C \in \underline{A}$ and $\vdash ?C :: \underline{A}$.

Tactic for ! :
$\vdash \underline{A}$ if $!C \in \underline{A}$ and $\vdash C :: (\underline{A} / !C)$ and $(\underline{A} / !C)$ is queried.

All the tactics of MALL are also tactics of CLL

The tactics of IMALL:

Tactic for 'Cut':
$A \vdash G$ if $\underline{A1} \vdash C$ and $\underline{A2} \vdash C^{\perp}$ (where $\underline{A1}, \underline{A2}$ is a partition of $\underline{A}$ and C is any formula).

Tactic for $\otimes$L:
$\underline{A} \vdash G$ if $C \otimes D \in \underline{A}$ and $C :: D :: (\underline{A} / C \otimes D) \vdash G$.

Tactic for &L1:
$\underline{A} \vdash G$ if $C \& D \in \underline{A}$ and $C :: (\underline{A} / C \& D) \vdash G$.

Tactic for &L2:
$\underline{A} \vdash G$ if $C \& D \in \underline{A}$ and $D :: (\underline{A} / C \& D) \vdash G$.

Tactic for $\oplus$L:
$\underline{A} \vdash G$ if $C \oplus D \in \underline{A}$ and $C :: (\underline{A} / C \oplus D) \vdash G$ and $D :: (\underline{A} / C \oplus D) \vdash G$.

Tactic for 1L:
$\underline{A} \vdash G$ if $1 \in \underline{A}$ and $(\underline{A} / 1) \vdash G$.

Tactic for $\multimap$L:
$\underline{A} \vdash G$ if $C \multimap D \in \underline{A}$ and $\underline{A1} \vdash C$ and $D :: \underline{A2} \vdash G$
(where $\underline{A1}$, $\underline{A2}$ is a partition of $(\underline{A} / C \multimap D)$).

Tactic for $\oplus$R1
$\underline{A} \vdash C \oplus D$ if $\underline{A} \vdash C$.

Tactic for $\oplus$R2
$\underline{A} \vdash C \oplus D$ if $\underline{A} \vdash D$.

Tactic for &R
$\underline{A} \vdash C \& D$ if $\underline{A} \vdash C$ and $\underline{A} \vdash D$.

Tactic for $\multimap$R
$\underline{A} \vdash C \multimap D$ if $C :: \underline{A} \vdash D$.

Tactic for $\otimes$R
$\underline{A} \vdash C \otimes D$ if $\underline{A1} \vdash C$ and $\underline{A2} \vdash D$ (where $\underline{A1}$, $\underline{A2}$ is a partition of $(\underline{A})$).

<u>Tactics of ILL:</u>

Tactic for W!:
$\underline{A} \vdash G$ if $!B \in \underline{A}$ and $(\underline{A} / !B) \vdash G$.

Tactic for D!:
$\underline{A} \vdash G$ if $!B \in \underline{A}$ and $B :: (\underline{A} / !B) \vdash G$.

Tactic for C!:
$\underline{A} \vdash G$ if $!C \in \underline{A}$ and $!C :: A \vdash G$.

Tactic for !R
$!\underline{A} \vdash !G$ if $!\underline{A} \vdash G$

All the tactics of IMALL are also tactics of ILL.

Also the tactics of ILinCLL are the same as those of CLL.
The tactics of ILinILL are the same as those of ILL.

Prolog implementation

The encoding of the tactics is shown below. These clauses will be interpreted by the interpreting layer described later. The use of an interpreting layer allows one to represent the tactics in as natural a form as possible, thus allowing easy verification and modification.

A clause of the form `'Logic1 is an extension of Logic2'` indicates that all the tactics of Logic2 are also tactics of Logic1. Thus tactics that belong to several logics do not have to be included several times.

```
% MALL

'MALL' tactic_of 'Cut'
converts ⊢ Problem into
⊢ A :: A1 and
⊢ A ⊥ :: A2
given Problem equ A1 plus A2 and A is_cut_formula.

'MALL' tactic_of '⊥'
converts ⊢ Prob where ⊥ ∈ Prob into
⊢ Prob - ⊥ .

'MALL' tactic_of ⊗
converts ⊢ Prob where C ⊗ D ∈ Prob into
⊢ C :: A1 and
⊢ D :: A2
given (Prob - C ⊗ D) equ A1 plus A2.

'MALL' tactic_of &
converts ⊢ Prob where C & D ∈ Prob into
⊢ [C / C & D]^Prob and
⊢ [D / C & D]^Prob.

'MALL' tactic_of ⅋
converts ⊢ Prob where C ⅋ D ∈ Prob into
⊢ [(C, D) / C ⅋ D]^Prob.

'MALL' tactic_of '2⊕'
converts ⊢ Prob where C ⊕ D ∈ Prob into
⊢ [D / C ⊕ D]^Prob.

'MALL' tactic_of '1⊕'
converts ⊢ Prob where C ⊕ D ∈ Prob into
⊢ [C / C ⊕ D]^Prob.

% CLL

'CLL' is_an_extension_of 'MALL'.

'CLL' tactic_of 'D?'
converts ⊢ Prob where ? C ∈ Prob into
⊢ [C / ? C]^Prob.

'CLL' tactic_of 'W?'
converts ⊢ Prob where ? C ∈ Prob into
⊢ Prob - ? C.
```

```
'CLL' tactic_of 'C?'
converts ⊢ Prob where ? C ∈ Prob into
⊢ ? C :: Prob.

'CLL' tactic_of !
converts ⊢ Prob where !C ∈ Prob into
⊢ [C / !C]^Prob
if check_ok(Prob).

% ILinCLL
'ILinCLL' is_an_extension_of 'CLL'.

% IMALL

'IMALL' tactic_of 'Cut'
converts A ⊢ G into
 A1 ⊢ C and
C :: A2 ⊢ G
given A equ A1 plus A2 and C is_cut_formula.

'IMALL' tactic_of '⊗L'
converts Prob ⊢ G where C ⊗ D ∈ Prob into
[(C, D) / C ⊗ D]^Prob ⊢ G.

'IMALL' tactic_of '⊸L'
converts A ⊢ G where C ⊸ D ∈ F into
A1 ⊢ C and
D :: A2 ⊢ G
given (A - C ⊸ D) equ A1 plus A2.

'IMALL' tactic_of '&L1'
converts Prob ⊢ G where C & D ∈ Prob into
[C / C & D]^Prob ⊢ G.

'IMALL' tactic_of '&L2'
converts Prob ⊢ G where C & D ∈ Prob into
[D / C & D]^Prob ⊢ G.

'IMALL' tactic_of '⊕L'
converts Prob ⊢ G where C ⊕ D ∈ Prob into
[C / C ⊕ D]^Prob ⊢ G and
[D / C ⊕ D]^Prob ⊢ G.

'IMALL' tactic_of '1L'
converts Prob ⊢ G where '1' ∈ Prob into
Prob - '1' ⊢ G.

'IMALL' tactic_of '⊸R'
converts A ⊢ C ⊸ D into
C :: A ⊢ D.

'IMALL' tactic_of '&R'
converts A ⊢ C & D into
A ⊢ C and A ⊢ D.

'IMALL' tactic_of '⊕R2'
converts A ⊢ C ⊕ D into
A ⊢ D.
```

```
'IMALL' tactic_of '⊕R1'
converts A ⊢ C ⊕ D into
A ⊢ C.

'IMALL' tactic_of '⊗R'
converts Context ⊢ C ⊗ D into
A1 ⊢ C and
A2 ⊢ D given
Context equ A1 plus A2.

% ILL

'ILL' is_an_extension_of 'IMALL'.

'ILL' tactic_of 'W!'
converts Prob ⊢ G where !C ∊ Prob into
Prob - !C ⊢ G.

'ILL' tactic_of 'D!'
converts Prob ⊢ G where !C ∊ Prob into
[C / !C]^Prob ⊢ G.

'ILL' tactic_of 'C!'
converts Prob ⊢ G where !C ∊ Prob into
!C :: Prob ⊢ G.

'ILL' tactic_of '!R'
converts A ⊢ !G into
A ⊢ G
if all_banged(A).

% ILinILL

'ILinILL' is_an_extension_of 'ILL'.

check_ok(Formulas) :-
remove_formula(!(_), Formulas, OtherFormulas),
queried(OtherFormulas).

queried([]).
queried([? _ | Rest]):-
      queried(Rest).

all_banged([]).
all_banged([! _ | Rest]):-
      all_banged(Rest).
```

A similar treatment is given to the Axioms:

```
% MALL

'MALL' axiom_of 'Identity' :
⊢ [A ⊥, A] is_axiomatic_if A is_an_atom.

'MALL' axiom_of 'Identity' :
⊢ [A, A ⊥] is_axiomatic_if A is_an_atom.

'MALL' axiom_of '1 axiom' :
⊢ ['1'] is_axiomatic.

'MALL' axiom_of 'T axiom' :
⊢ G is_axiomatic_if 'T' ∊ G.

% IMALL

'IMALL' axiom_of 'T axiom' :
A ⊢ 'T' is_axiomatic.

'IMALL' axiom_of 'Identity' :
[G] ⊢ G is_axiomatic_if G is_an_atom.

'IMALL' axiom_of '0 axiom' :
F ⊢ G is_axiomatic_if '0' ∊ F.

'IMALL' axiom_of '1 axiom' :
[] ⊢ '1' is_axiomatic.
```

The syntax of an axiom is:

```
Axiom ::=          Logic axiom_of Axiom_Name:
                   Problem is_axiomatic.
                   | Logic axiom_of Axiom_Name:
                   Problem is_axiomatic_if Condition
```

And the syntax of a tactic is:

```
Tactic ::=         Logic   tactic_of  Tactic_Name
                   converts  Problem  [ where  Condition ]  into
                   NProblem_List [ if Side_Condition ] .
```

where

```
Problem ::=        List ⊢ Formula
                   |  ⊢ List

NProblem ::=       NList ⊢ Formula
                   |  ⊢ NList

NList ::=          Formula ::  NList
                   | [ Formula / Formula ] ^ NList
                   | [ (Formula, Formula) / Formula ] ^ NList
                   | Formula
```

NProblem_List ::= **NProblem** {and **NProblem** }
| **NProblem** and
NProblem
given **Natural_List** equ **Natural_List** plus **Natural_List**
[and **Formula** is_cut_formula]

Logic ::= Prolog String

Tactic_Name ::= Prolog String

Axiom_Name ::= Prolog String

Condition ::= **Formula** ∊ **NList**
| **Formula** is_an_atom

Formula ::= Prolog Term

Formula ::= Prolog Term

Side_Condition ::= Prolog Clause

List ::= Prolog List

Interpreter for axioms

```
get_axiom(Logic, Axiom, Problem, Condition) :-
      Logic axiom_of Axiom : Problem is_axiomatic_if Condition.

get_axiom(Logic, Axiom, Problem, true) :-
      Logic axiom_of Axiom : Problem is_axiomatic.

get_axiom(Logic, Axiom, Problem, Condition) :-
      Logic1 axiom_of Axiom : Problem is_axiomatic_if Condition,
      Logic is_an_extension_of Logic1.
```

The predicate get_axiom returns information about an axiom that may be applicable to Problem in Logic.

```
% trivial(+Logic, +Problem, -Name)
trivial(Logic, Problem, Name) :-
      get_axiom(Logic, Name, Problem, Condition),
      Condition, !.

A is_an_atom   :-      atom(A).
A ∊ B                  :-       on(A, B).
```

Thus the predicate trivial succeeds if Problem is trivial in Logic, and returns the name of the axiom by which it is trivial.

Interpreter for tactics

```
get_tactic(Logic, Tactic, Problem, Condition, Sprobs , SideCondition) :-
      Logic tactic_of Tactic converts Problem where Condition into Sprobs if SideCondition.

get_tactic(Logic, Tactic, Problem, true, Sprobs , SideCondition) :-
      Logic tactic_of Tactic converts Problem into Sprobs if SideCondition.
```

```
get_tactic(Logic, Tactic, Problem, Condition, NewProbs, true) :-
      Logic tactic_of Tactic converts Problem where Condition into NewProbs.

get_tactic(Logic, Tactic, Problem, true, Sprobs , true) :-
      Logic tactic_of Tactic converts Problem into Sprobs.

get_tactic(Logic, Tactic, Problem, Condition, Sprobs , SideCondition) :-
      Logic is_an_extension_of Logic1,
      get_tactic(Logic1, Tactic, Problem, Condition, Sprobs , SideCondition).
```

The predicate get_tactic returns information about tactics that may be applicable to Problem in Logic. It takes into account the various possible forms of a tactic.

The predicate evaluate_list transforms a NList into a Prolog List. Items that have yet to be defined in the Tactic definition are left uninstantiated in the Prolog List:

```
evaluate_list(A, A) :- var(A), !.          % Catches variables and leaves them uninstantiated
evaluate_list([(A, B)/C]^List, Result) :- !,
      evaluate_list(List, NewList),
      replace_formula(C, [A, B], NewList, Result).
evaluate_list([A/C]^List, Result) :- !,
      evaluate_list(List, NewList),
      replace_formula(C, [A], NewList, Result).
evaluate_list(List-Formula, Result) :- !,
      evaluate_list(List, NewList),
      remove_formula(Formula, NewList, Result).
evaluate_list(A::L, [A | NL]) :- !,
      evaluate_list(L, NL).
evaluate_list(A, A).

replace_formula(_, _, [], []).
replace_formula(Formula, NewFormulas, [Formula | RestList], NewList) :- !,
      append(NewFormulas, RestList, NewList).
replace_formula(Formula, NewFormulas, [First | Rest], [First | NewRest]) :-
      replace_formula(Formula, NewFormulas, Rest, NewRest).

remove_formula(Formula, Prob, NProb) :-
      one(remove(Formula, Prob, NProb)).
```

The predicate evaluate_sequent transforms a NProblem into a Problem:

```
evaluate_sequent(⊢A, ⊢NA) :- !,
      evaluate_list(A, NA).
evaluate_sequent(A⊢G, NA⊢G) :-
      evaluate_list(A, NA).
```

The predicate choose_formula_satisfies allows the user to select a formula A which fulfills the Condition A ∈ B:

```
choose_formula_satisfies(true).
choose_formula_satisfies(∈(Template, Formulae)) :-
      find_suitable(Template, Formulae, Suitable_Formulae),
      (Suitable_Formulae = [Template]          % Instantiates Template to
                                                % the one suitable formula
      ;select_one(Suitable_Formulae, Template)), !.   % Allows user
                                                          % to select a formula
```

```
find_suitable(Template, Formulae, Suitable) :-
      setof(Template, (on(Template, Formulae)), Suitable).
```

The Tactics are used for two purposes. The first is to apply a tactic to a problem, giving subproblems. The second is to form a list of all the tactics of a particular logic and those which can be applied to a particular problem.

The code for applying a tactic to a problem is shown below:

```
apply(Logic, Tactic, Problem, NewProblems) :-
      get_tactic(Logic, Tactic, Problem, Condition, Problems, _),
      choose_formula_satisfies(Condition), !,
      interpret(Problems, NewProblems), !.

apply(Logic, Axiom, Problem, []) :-
      axioms_as_tactics_set,  % Axioms can be applied as tactics
      get_axiom(Logic, Axiom, Problem, Condition),
      call(Condition), !.

interpret(A and B, Problems) :-
      evaluate_sequent(A, Problem1),
      evaluate_sequent(B, Problem2),
      append([Problem1], [Problem2], Problems).
interpret(A and B given List equ List1 plus List2 and C is_cut_formula, [Problem1,
Problem2]) :-
      evaluate_list(List, NList),
      evaluate_sequent(A, Problem1),
      evaluate_sequent(B, Problem2),
      perform_cut(NList, Problem1, Problem2, List1, List2, C).
interpret(A and B given List equ List1 plus List2, [Problem1, Problem2]) :-
      evaluate_list(List, NList),
      evaluate_sequent(A, Problem1),
      evaluate_sequent(B, Problem2),
      split_cont(NList, Problem1, Problem2, List1, List2).
interpret(Problem, [NewProblem]) :-
      evaluate_sequent(Problem, NewProblem)
```

The code for determining all the tactics of a Logic, and all those which are applicable to a particular problem, is shown below:

```
get_tactics(Logic, Problem, Applicable, All) :-
      all_tactics(Logic, All),
      applicable_tactics(Logic, Problem, Applicable).

all_tactics(Logic, Tactics) :-
      bagof(Tactic, (get_one_tactic(Logic, Tactic)), Tactics), !.
all_tactics(_, []).

get_one_tactic(Logic, Tactic) :-
      get_tactic(Logic, Tactic, _, _, _, _).
get_one_tactic(Logic, Axiom) :-
      axioms_as_tactics_set,  % Axioms may be applied as tactics
      get_axiom(Logic, Axiom, _, _).
```

```
applicable_tactics(Logic, Problem, Tactics) :-
      bagof(Tactic, (get_one_applicable_tactic(Logic, Problem, Tactic)), Tactics), !.
applicable_tactics(_, _, []).

get_one_applicable_tactic(Logic, Prob, Tact) :-
      get_tactic(Logic, Tactic, Prob, Condition, _, SideCondition),
      Condition,
      Side_Condition.

get_one_applicable_tactic(Logic, Problem, Name) :-
      axioms_as_tactics_set,
      get_axiom(Logic, Name, Problem, Conditions),
      Conditions.
```

Predicates Used:

```
split_context(+Context, +Prob1, +Prob2, -Context1, -Context2)
```

Context	: List of Formulae
Prob1	: Problem
Prob2	: Problem
Context1	: List of Formulae
Context2	: List of Formulae

The user is presented with a scrolling menu containing all the formulae in Context, and asked to select all those that he wishes to form part of the first problem. These are returned as Context1. The rest are returned as Context2. Prob1 and Prob2 are used to inform the user on the appearance of the subproblems that he is creating, to help him in his choice.

```
perform_cut(+Context, +Prob1, +Prob2, -Context2, -Context1, -Cut)
```

Context	: List of Formulae
Prob1	: Problem
Prob2	: Problem
Context1	: List of Formulae
Context2	: List of Formulae
Cut	: Formula

The user is presented with a scrolling menu containing all the formulae in Context, and asked to select all those that he wishes to form part of the first problem, and enter the cut formula C. Prob1 and Prob2 are used to inform the user on the appearance of the subproblems that he is creating, to help him in his choice.

These predicates fail upon backtracking.

2.5 Implementation of Transformations

Negation Handling

To ensure that a tactic can be applied to every non atomic formula within a sequent, we need to ensure that linear negation and linear implication are never the main connective of any sequent ($A \multimap B$ is just a meta-notation for $A^{\perp} \parr B$). Thus negations are 'moved inwards' according to the equivalences:

$$\begin{array}{lcl}
(A \otimes B)^{\perp} & = & A^{\perp} \parr B^{\perp} \\
(A \parr B)^{\perp} & = & A^{\perp} \otimes B^{\perp} \\
(A \,\&\, B)^{\perp} & = & A^{\perp} \oplus B^{\perp} \\
(A \oplus B)^{\perp} & = & A^{\perp} \,\&\, B^{\perp} \\
1^{\perp} & = & \perp \\
\perp^{\perp} & = & 1 \\
\top^{\perp} & = & 0 \\
0^{\perp} & = & \top
\end{array}$$

Linear implications are removed by applying the equivalence:

$$A \multimap B \quad = \quad A^{\perp} \parr B$$

The above transformations are not defined recursively. They simply translate a formula into a formula to which a tactic may be applied. Care must be taken to ensure that a subformula of the form $A^{\perp\perp}$ is never created, when applying the transformations as $A^{\perp\perp} = A$. Though formulae of the form $A^{\perp\perp}$ are syntactically well-formed, they are best avoided for clarity.

Prolog implementation

```
transform(⊢ A, ⊢ B) :- map(transform_cll_formula, A, B).
transform_cll_formula(Formula⊥, NewFormula) :- !,
      negate(Formula, NewFormula).
transform_cll_formula(A⊥ ⊸ B, (A ⅋ B)) :- !.
transform_cll_formula(A ⊸ B, ((A⊥) ⅋ B)) :- !.
transform_cll_formula(A, A).
```

The predicate negate is used to take advantage of Prologs' first argument indexing.

```
negate('1', '⊥') :- !.
negate('⊥', '1') :- !.
negate('T', '0') :- !.
negate('0', 'T') :- !.

negate((A' ⊗ B⊥), A ⅋ B) :- !.
negate((A⊥ ⊗ B), A ⅋ B⊥) :- !.
negate((A ⊗ B⊥), A⊥ ⅋ B) :- !.
negate((A ⊗ B), A⊥ ⅋ B⊥) :- !.

negate((A⊥ & B⊥), A ⊕ B) :- !.
negate((A⊥ & B), A ⊕ B⊥) :- !.
negate((A & B⊥), A⊥ ⊕ B) :- !.
negate((A & B), A⊥ ⊕ B⊥) :- !.
negate((!A⊥), ?(A)) :- !.
negate((!A), ?(A⊥)) :- !.

negate((?A⊥), !(A)) :- !.
negate((?A), !(A⊥)) :- !.

negate((A⊥ ⅋ B⊥), A ⊗ B) :- !.
negate((A⊥ ⅋ B), A ⊗ B⊥) :- !.
negate((A ⅋ B⊥), A⊥ ⊗ B) :- !.
negate((A ⅋ B), A⊥ ⊗ B⊥) :- !.

negate((A⊥ ⊕ B⊥), A & B) :- !.
negate((A⊥ ⊕ B), A & B⊥) :- !.
negate((A ⊕ B⊥), A⊥ & B) :- !.
negate((A ⊕ B), A⊥ & B⊥) :- !.

negate((A ⊸ B⊥), A ⊗ B) :- !.
negate((A ⊸ B), A ⊗ B⊥) :- !.

negate(((A)⊥), A) :- !.
negate(A, A⊥) :- atom(A), !.
```

Transforming IL to LL

When working in ILinCLL and ILinILL, IL Formulae are 'marked' as such by being enclosed in brackets (⟦⟧). The transformations from IL to LL are given below. These ensure that a LL Tactic can be applied to the formula after transformation.

$[\![A \,\&\, B]\!]$	=	$[\![A]\!] \,\&\, [\![B]\!]$	
$[\![A \vee B]\!]$	=	$![\![A]\!] \oplus ![\![B]\!]$	
$[\![A \rightarrow B]\!]$	=	$![\![A]\!] \multimap [\![B]\!]$	$= ?[\![A]\!]^{\perp} ⅋ [\![B]\!]$
$[\![\sim A]\!]$	=	$![\![A]\!] \multimap 0$	$= ?[\![A]\!]^{\perp} ⅋ 0$
$[\![\wedge]\!]$	=	0	
$[\![a]\!]$	=	a	

Linearly negated IL formula must both be transformed, and have negation moved inwards:

$[\![A \,\&\, B]\!]^{\perp}$	=	$([\![A]\!] \,\&\, [\![B]\!])^{\perp}$	$= [\![A]\!]^{\perp} \oplus [\![B]\!]^{\perp}$
$[\![A \vee B]\!]^{\perp}$	=	$(![\![A]\!] \oplus ![\![B]\!])^{\perp}$	$= ?[\![A]\!]^{\perp} \,\&\, ?[\![B]\!]^{\perp}$
$[\![A \rightarrow B]\!]^{\perp}$	=	$(![\![A]\!] \multimap [\![B]\!])^{\perp}$	$= ![\![A]\!] \otimes [\![B]\!]^{\perp}$
$[\![\sim A]\!]^{\perp}$	=	$(![\![A]\!] \multimap 0)^{\perp}$	$= ![\![A]\!] \otimes \top$
$[\![\wedge]\!]^{\perp}$	=	$\top$	
$[\![a]\!]^{\perp}$	=	$a^{\perp}$	

Prolog implementation:

```
transform_cll_formula((⟦ A ⟧) ⊥, NewA1) :- !,
     translate_and_negate_il_cll(A, NewA1).
transform_cll_formula(⟦A⟧, NewA) :- !,
     translate_il_ll(A, NewA).
transform_cll_formula(A, A).
```

```
translate_il_ll(A & B, TA & TB) :-
      mark_il(A, TA),
      mark_il(B, TB).
translate_il_ll(A∨B, !TA ⊕ !TB) :-
      mark_il(A, TA),
      mark_il(B, TB).
translate_il_ll(A → B, !TA ⊗ TB) :-
      mark_il(A, TA),
      mark_il(B, TB).
translate_il_ll(¬A, !TA ⊗ 'T') :-
      mark_il(A, TA).
translate_il_ll(∧, '0').
translate_il_ll(A, A ) :- atom(A).

translate_and_negate_il_cll(F, NF) :-
      translate_il_ll(F, F1),
      negate(F1, NF).

mark_il(A, A) :- atomic(A).
mark_il(A, ⟦A⟧).

% ILinILL translation
translate(Problems, Newproblems) :-
      map(translate_problem, Problems, Newproblems).

translate_problem(A ⊢ G, NA ⊢ NG) :-
      map(translate_il_ill,A, NA),
      translate_il_ill(G, NG).

translate_il_ill(⟦A⟧, NA) :- !,
      translate_il_ll(A, NA).
translate_il_ill(A,A).
```

The predicate mark_il is used to ensure that a subformula of the form ⟦∧⟧, or ⟦A⟧ where A is an atom, is never created. It does this by effectively translating atomic subformulae immediately rather than marking them to be transformed latter. This is done for clarity.

Transformations are applied to a sequent by a call to transform_problems.

```
transformation('MALL', transform).
transformation('CLL', transform).
transformation('ILinCLL', transform).
transformation('ILinILL', translate).
% (No transformations need to be applied to a ILL or IMALL sequent)

transform_problems(Logic, Problems, Newproblems) :-
      transformation(Logic, Op), !,
      Op(Problems, Newproblems).
transform_problems(_, Problems, Problems) :- !.
```

Initial Transformations

CLL and MALL problems entered as two sided sequents need to be transformed to one sided sequents. This is done by change_to_onesided_sequent:

```
change_to_onesided_sequent(F ⊢ G , ⊢ Newproblem) :-
  map(negate , F ,X) ,
  append(X , G , Newproblem ) .
```

In the case of an IL sequent all the formulae need to be marked as IL formulae, those on the left being queried or banged:

```
translate_il_cll(Assums ⊢ G , ⊢ Newproblem) :-
  map(mark_il, G, NewGoal) ,
  map(mark_il, Assums, NAS) ,
  map(assumption_query , NAS , NewAssums) ,
  append(NewAssums , NewGoal , Newproblem1 ).

translate_il_ill(Assumtions ⊢ Goal, NewAssumtions ⊢ NewGoal) :-
      mark_il(Goal, NewGoal),
      map(mark_il, Assumtions, Assumtions1),
      map(assumption_bang, Assumtions1, NewAssumtions).

assumption_query(Formula, ? Formula ⊥).
assumption_bang(Formula, ! Formula).
```

Initial transformations are applied to a sequent by a call to initial_transform.

```
initial_transformation('CLL', change_to_onesided_sequent).
initial_transformation('MALL', change_to_onesided_sequent).
initial_transformation('ILinCLL', translate_il_cll).
initial_transformation('ILinILL', translate_il_ill).
% (No initial transformations need to be applied to a ILL or IMALL sequent)

initial_transform(Logic, Problem, NewProblem) :-
      initial_transformation(Logic, Op), !,
      Op(Problem, NewProblem), !.
initial_transform(_, Problem, Problem).
```

2.6 The Proof Editor

The code described above allow one to:

A) Transform an entered problem into a standard form.

B) Transform a problem into an equivalent problem to which a tactic can be applied.

C) Apply a tactic to a problem to give subproblems.

D) Determine if a problem is trivial.

All the above are specific to each logic and are therefore abstracted from the interactive proof editing mechanism shown below. This was originally developed for use in MacLogic [5]:

```
solve(Window, Logic, _Depth, Problem) :-
      trivial(Logic, Problem, Name), !,
      write_bf( Window,'■' ) ,
      find_next_prob( Window ) .

solve(Window, Logic, Depth, Problem) :-
      repeat,
      get_applicable_tactic(Problem, Logic, Tactic),
      (is_backtrack(Tactic) -> !, fail
      ;true
      ),
      apply(Tactic, Problem, Subproblems),
      transform_problems(Logic, Subproblems, NewSubproblems),
      maybe_check(Logic, NewSubproblems),
      proof_write(Window, Tactic, Depth, NewSubproblems),
      NewDepth is Depth + 1,
      solve_problems(Window, Logic, NewDepth, NewSubproblems).

solve_problems(_Window, _Logic, _Depth, []) :- !.
solve_problems(Window, Logic, Depth, [Problem | OtherProblems]) :-
      solve(Window, Logic, Depth, Problem),
      solve_problems(Window, Logic, Depth, OtherProblems).
```

This is a recursive definition of a tree, with information on subtrees of a branch provided by the predicate `apply`, and information on end branches provided by the predicate `trivial`. The repeat-fail loop is not standard and is designed to allow the user to 'backtrack' through the tree one step at a time.

The predicate `get_applicable_tactic` allows the user to select a tactic. The user may only select a tactic which is applicable to the current problem.

Output

While completing a proof, the user may at any stage decide to undo a step. In Prolog all input and output is deterministic in that (while backtracking) they can not be resatisfied. Thus, upon backtracking, a call to write will fail, but leave the written text upon the screen. This is desirable in most cases (eg writing out all solutions to a problem). What is needed for interactive input/output is a version of write which, if backtracked on, will fail and 'unwrite' what had previously been written. This is done by creating a backtrackable version of write, cursor and nl. This technique is due to Richard O'Keefe.

```
write_bf( Window , Text ):-
                              cursor( Window , From , To ),
                              wsltxt( Window , From , To , Oldtext ),
                              write( Window , Text ),
                              cursor( Window , _ , Newto ),
                              asserta( written( Oldtext , Newto , From , To ) ).
write_bf( Window , Text ):-
                              retract( written( Oldtext , Newto , From , To ) ),
                              cursor( Window , From , Newto ),
                              write( Window , Oldtext ),
                               cursor( Window , From , To ), !,
                              fail.

cursor_bf( Window , From , To) :-
                              cursor( Window , Oldfrom , Oldto ),
                              cursor( Window , From , To ),
                              asserta( cursored( Oldfrom , Oldto ) ).
cursor_bf( Window , From , To ):-
                              retract( cursored( Oldfrom , Oldto ) ),
                              cursor( Window , Oldfrom , Oldto ), !,
                              fail.

nl_bf( Window ) :-
  write_bf( Window, '~M' ) .
```

2.7 Implementation of 'IILinIMALL'

A variant of Implicative Intuitionistic Logic can be embedded in IMALL [11]. This embedding shall be called IILinIMALL. This is easily implemented by the following code. As with the other enbeddings, IL sequents are marked with square brackets, but as this is an asymmetrical transformation, they also need to be marked by either + or -.

IILinIMALLcan be considered to be an extension of IMALL:

```
initial_sequent_type('IILinIMALL', list ⊦ formula).
initial_syntax('IILinIMALL', 'IIL').

'IILinIMALL' is_an_extension_of 'MALL'.
'IILinIMALL' is_a_syntactic_extension_of 'MALL'.
'IILinIMALL' is_a_syntactic_extension_of 'IIL'.

symbol_for("⟦", 'IILinIMALL').
symbol_for("⟧", 'IILinIMALL').
symbol_for("k", 'IILinIMALL').
symbol_for("⊥", 'IILinIMALL').
symbol_for("+", 'IILinIMALL').
symbol_for("-", 'IILinIMALL').

symbol_for("→", 'IIL').
symbol_for("⋀", 'IIL').
symbol_for("⌄", 'IIL').
symbol_for("~", 'IIL').
```

The initial transformations are shown below. Initially, depth reduction must be applied to the problem, and any negations and disjunctions transformed into implicative formulae:

```
initial_transformation("IILinIMALL'', translate_il_imall).

translate_il_imall(Problem, NewProblem) :-
  implicize_sequent(Problem, Problem1) ,
  depth_reduce(Problem1, Problem2),
  mark_ill(Problem2, NewProblem).

mark_ill(A ⊦ G, [k | NA] ⊦ ⟦G⟧+) :-
      map(mark_ill_formula, A, NA).

mark_ill_formula(A, ⟦(A)⟧-).

implicize_sequent(A ⊦ G, NA ⊦ NG) :-
  map(implicize_formula, A, NA),
  implicize_formula(G, NG).

implicize_formula(A → B, NA → NB) :- !,
  implicize_formula(A, NA),
  implicize_formula(B, NB) .
implicize_formula(A ⌄ B, (NB → NA) → NA) :- !,
  implicize_formula(A, NA),
  implicize_formula(B, NB) .
implicize_formula(¬ A,  NA → ⋀) :- !,
  implicize_formula(A, NA) .
implicize_formula(A, A) .
```

```
depth_reduce(As ⊦ G, NA ⊦ NG) :-
   remove((A→B)→(C→D), As, NA1),
   depth_reduce(['X'→(C→D), (A→B)→'X' | NA1] ⊦ G, NA ⊦ NG) .
depth_reduce(As ⊦ G, NA ⊦ NG) :-
   remove(P→((A→B)→C), As, NA1), atom(P),
   depth_reduce([P→('X'→C), (A→B)→'X' | NA1] ⊦ G, NA ⊦ NG) .
depth_reduce(As ⊦ G, NA ⊦ NG) :-
   remove(P→(A→(B →C)), As, NA1), atom(P),
   depth_reduce(['X'→(B→C), P→(A→'X') | NA1] ⊦ G, NA ⊦ NG) .
depth_reduce(As ⊦ G, NA ⊦ NG) :-
   remove(((A→B)→C)→P, As, NA1), atom(P),
   depth_reduce(['X'→(A→B), ('X'→C)→P | NA1] ⊦ G, NA ⊦ NG) .
depth_reduce(As ⊦ G, NA ⊦ NG) :-
   remove((A→(B→C))→P, As, NA1), atom(P),
   depth_reduce([(B→C)→'X', (A→'X')→P | NA1] ⊦ G, NA ⊦ NG) .
depth_reduce(As ⊦ (A→B)→(C→D), NA ⊦ NG) :-
   depth_reduce([(C→D)→'X' | As] ⊦ (A→B)→'X', NA ⊦ NG) .
depth_reduce(As ⊦  P→(A→(B →C)), NA ⊦ NG) :-
  atom(P),
   depth_reduce([(B→C)→'X' | As] ⊦ P→(A→'X'), NA ⊦ NG) .
depth_reduce(As ⊦  P→((A→B)→C), NA ⊦ NG) :-
  atom(P),
   depth_reduce(['X'→(B→C)  | As] ⊦ P→('X'→C), NA ⊦ NG) .
depth_reduce(As ⊦ (A→(B→C))→P, NA ⊦ NG) :-
  atom(P),
   depth_reduce(['X'→(B→C)  | As] ⊦ (A→'X')→P, NA ⊦ NG) .
depth_reduce(As ⊦ ((A→B)→C)→P, NA ⊦ NG) :-
  atom(P),
   depth_reduce([(B→C)→'X'  | As] ⊦ ('X'→C)→P, NA ⊦ NG) .
depth_reduce(A, A) .
```

The transformations are shown below:

```
transformation("IILinIMALL' ', astransform_problems).

astransform_problems(Problems, NewProblems) :-
  map(astransform_sequent, Problems, NewProblems).

astransform_sequent(A ⊦ B, NA ⊦ NB) :-
  map(astransform, A, NA),
  astransform(B, NB ).

astransform(⟦A→B⟧+, k⊗(⟦A⟧- ⊸(k⊸⟦B⟧+)))  :- !.
astransform(⟦A⟧+, k⊗(NA⊕'⊥'⊗'T')) :- atom_transform(A , NA),! .

astransform(⟦(A→B)→C⟧-,k⊸((((⟦B→C⟧- ⊸(k⊸⟦A→B⟧+))⊸(k⊗'⊥'))⊕(k⊗⟦C⟧-)))  :- !.
astransform(⟦P→A⟧-, k⊸((k⊸(⟦P⟧+)⊸(k⊗'⊥'))⊕(k⊗⟦A⟧-)))  :-
  atom(P), !.
astransform(⟦A⟧-, NA) :- atom_transform(A, NA), !.

astransform(P, P).

atom_transform(⋀ , '⊥') :- !.
atom_transform(A , A) :- atom(A) .
```

3 THE THEOREM PROVERS

3.1 IMALL Theorem Prover

It has been established that IMALL is decidable and PSPACE-complete [10]. The following IMALL theorem prover is a simple implementation of the tactics of IMALL.

Heuristics are built into the theorem prover by the ordering of the clauses. This assigns priorities to the various tactics. Axioms have the highest priority, followed by invertible tactics, followed by the non-invertible tactics.

For efficiency the Identity axiom is coded to check for pairings between formulae and not just atoms. This is justified by lemma 1.1, in Chapter 4.

```
prove_imall0([A] ⊢ A)  :- !.

prove_imall0([] ⊢ '1')  :- !.

prove_imall0(_ ⊢ 'T')  :- !.

prove_imall0(A ⊢ _)     :- !,
      on('0', A), !.

prove_imall0(A ⊢ C&D) :- !,     % Lemma 1.11
      prove_imall0(A ⊢ C), !,
      prove_imall0(A ⊢ D), !.

prove_imall0(A ⊢ C⊸D) :- !,     % Lemma 1.16
      prove_imall0([C | A] ⊢ D), !.

prove_imall0(A ⊢ G) :-
      remove('1', A, NA), !,    % Lemma 1.13
      prove_imall0(NA ⊢ G), !.

prove_imall0(A ⊢ G) :-
      remove(C⊗D, A, NA), !, % Lemma 1.12
      prove_imall0([C, D | NA] ⊢ G), !.

prove_imall0(A ⊢ G) :-
      remove(C⊕D, A, NA), !, % Lemma 1.14
      prove_imall0([C | NA] ⊢ G), !,
      prove_imall0([D | NA] ⊢ G), !.

prove_imall0(A ⊢ G) :-
      remove(C&D, A, NA),
      (prove_imall0([C | NA] ⊢ G)
      ;prove_imall0([D | NA] ⊢ G)
      ), !.

prove_imall0(A ⊢ C⊕D) :-
      ( prove_imall0(A ⊢ C)
      ; prove_imall0(A ⊢ D)
      ), !.

prove_imall0(A ⊢ C⊗D) :-
      partition(A, A1, A2),
      prove_both(A1 ⊢ C, A2 ⊢ D), !.
```

```
prove_imall0(A ⊢ G) :-
      remove(C⊸D, A, NA),
      partition(NA, A1, A2),
      prove_both(A1 ⊢ C, [D | A2] ⊢ G), !.

prove_both(P1, P2) :-
      prove_imall0(P1), !,       % Prevents multiple proofs
      prove_imall0(P2), !.


partition([], [], []).
partition([F | R], [F | R1], R2) :-
      partition(R, R1, R2).
partition([F | R], R1, [F | R2]) :-
      partition(R, R1, R2).
```

Cuts are used to improve efficiency. If an invertible tactic is applied to a problem and one or more of the new subproblems are unsolvable, then the problem is unsolvable. There is thus no point in attempting to solve the problem by applying an alternative tactic to it. This fact is expressed by a cut at the appropriate place. The proofs of invertibility are given in Chapter 4.

Elsewhere cuts are used to prevent the theorem prover from generating multiple proofs. Some problems are capable of being solved in different ways, e.g., $\top \vdash \top$ by the tactics of I and $\top$. If for example prove_both is called upon to solve two problems, if the second problem turns out to be unsolvable then there is no point in backtracking to find an alternative solution to the first problem. Thus a cut is inserted to prevent this from happening.

The above theorem prover is however extremely inefficient. The reason for this is that the tactics of $\otimes$R and $\multimap$L require the partitioning of a context.

The predicate partition is repeatedly called upon to generate new partitions of a context until a suitable one is found. Generating all possible partitions of formulae takes time exponential in the number of formulae.

Dealing with the context splitting tactics:

Hodas and Miller [9] showed that the need to split contexts in order to prove multiplicative conjunctions on the right can be handled by viewing the proof search as a process that takes a context, consumes part of it, and returns the rest to be consumed elsewhere.

For this purpose we develop a new calculus IMALL*:

An IMALL* sequent has the form $\{\underline{B}\}\text{-}\{\underline{U}\} \vdash G$ where $\underline{B}$ and $\underline{U}$ are contexts and $\underline{U}$ is a sub-multiset of $\underline{B}$. $\underline{U}$ is called the **unused context**, and $\underline{B}$ is called the **input context.**

The IMALL* sequent $\{\underline{B}\}\text{-}\{\underline{U}\} \vdash G$ is equivalent to the IMALL sequent $\underline{E} \vdash G$ where $\underline{E}$ is the context obtained by removing one occurrence of each formula in $\underline{U}$ from $\underline{B}$. This is always possible if $\underline{U}$ is a sub-multiset of $\underline{B}$.

The IMALL sequent $\underline{A} \vdash G$ is equivalent to the IMALL* sequent $\{\underline{A}\}\text{-}\{\} \vdash G$.

$$\frac{\underline{A1} \vdash C \qquad \underline{A2} \vdash D}{\underline{A1}, \underline{A2} \vdash C \otimes D}\ \otimes R \qquad \text{becomes} \qquad \frac{\{\underline{A}\}\text{-}\{\underline{U1}\} \vdash C \qquad \{\underline{U1}\}\text{-}\{\underline{U}\} \vdash D}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \otimes D}\ \otimes R^*$$

$$\begin{array}{c}\underline{A1} \vdash C \quad D, \underline{A2} \vdash G \\ \hline C \multimap D, \underline{A1}, \underline{A2} \vdash G\end{array} \multimap L \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}\}\text{-}\{\underline{U1}\} \vdash C \quad \{\underline{U1}, D\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A} \\ \hline \{C \multimap D, \underline{A}\}\text{-}\{\underline{U}\} \vdash G\end{array} \multimap L^*$$

$$\begin{array}{c}\underline{A}, C \vdash D \\ \hline \underline{A} \vdash C \multimap D\end{array} \multimap R \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}, C\}\text{-}\{\underline{U}\} \vdash D \quad \underline{U} \subseteq \underline{A} \\ \hline \{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \multimap D\end{array} \multimap R^*$$

$$\begin{array}{c}\underline{A}, C \vdash G \\ \hline \underline{A}, C\&D \vdash G\end{array} \&L1 \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}, C\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A} \\ \hline \{\underline{A}, C\&D\}\text{-}\{\underline{U}\} \vdash G\end{array} \&L1^*$$

$$\begin{array}{c}\underline{A}, D \vdash G \\ \hline \underline{A}, C\&D \vdash G\end{array} \&L2 \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}, D\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A} \\ \hline \{\underline{A}, C\&D\}\text{-}\{\underline{U}\} \vdash G\end{array} \&L2^*$$

$$\begin{array}{c}\underline{A} \vdash C \quad \underline{A} \vdash D \\ \hline\hline \underline{A} \vdash C\&D\end{array} \&R \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \quad \{\underline{A}\}\text{-}\{\underline{U}\} \vdash D \\ \hline\hline \{\underline{A}\}\text{-}\{\underline{U}\} \vdash C\&D\end{array} \&R^*$$

$$\begin{array}{c}\underline{A} \vdash C \\ \hline \underline{A} \vdash C \oplus D\end{array} \oplus R1 \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \\ \hline \{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \oplus D\end{array} \oplus R1^*$$

$$\begin{array}{c}\underline{A} \vdash D \\ \hline \underline{A} \vdash C \oplus D\end{array} \oplus R2 \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}\}\text{-}\{\underline{U}\} \vdash D \\ \hline \{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \oplus D\end{array} \oplus R2^*$$

$$\begin{array}{c}\underline{A} \vdash G \\ \hline\hline \underline{A}, 1 \vdash \quad G\end{array} 1L \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}\}\text{-}\{\underline{U}\} \vdash G \\ \hline\hline \{\underline{A}, 1\}\text{-}\{\underline{U}\} \vdash G\end{array} 1L^*$$

$$\begin{array}{c}\underline{A}, C \vdash G \quad \underline{A}, D \vdash G \\ \hline\hline \underline{A}, C \oplus D \vdash G\end{array} \oplus L \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}, C\}\text{-}\{\underline{U}\} \vdash G \quad \{\underline{A}, D\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A} \\ \hline\hline \{\underline{A}, C \oplus D\}\text{-}\{\underline{U}\} \vdash G\end{array} \oplus L^*$$

$$\begin{array}{c}\underline{A}, C, D \vdash G \\ \hline\hline \underline{A}, C \otimes D \vdash G\end{array} \otimes L \quad \text{becomes} \quad \begin{array}{c}\{\underline{A}, C, D\}\text{-}\{\underline{U}\} \vdash G \\ \hline\hline \{\underline{A}, C \otimes D\}\text{-}\{\underline{U}\} \vdash G\end{array} \otimes L^*$$

$$\begin{array}{c} \\ \hline G \vdash G\end{array} I \quad \text{becomes} \quad \begin{array}{c} \\ \hline \{G, \underline{U}\}\text{-}\{\underline{U}\} \vdash G\end{array} I^*$$

$$\begin{array}{c} \\ \hline \vdash 1\end{array} 1 \quad \text{becomes} \quad \begin{array}{c} \\ \hline \{\underline{A}\}\text{-}\{\underline{A}\} \vdash 1\end{array} 1^*$$

$$\begin{array}{c} \\ \hline \underline{A}, 0 \vdash G\end{array} 0 \quad \text{becomes} \quad \begin{array}{c} \\ \hline \{\underline{A}, 0\}\text{-}\{\underline{U}\} \vdash G\end{array} 0^* \ (\underline{U} \text{ is a subset of } \underline{A})$$

$$\begin{array}{c} \\ \hline \underline{A} \vdash T\end{array} T \quad \text{becomes} \quad \begin{array}{c} \\ \hline \{\underline{A}\}\text{-}\{\underline{U}\} \vdash T\end{array} T^* \ (\underline{U} \text{ is a subset of } \underline{A})$$

All these tactics preserve the characteristic of IMALL* sequents that the unused context is a submulti-set of the input context, ie if the sequents above the line have this property then so to do those below the line.

Lemma 2.1: The system of IMALL* is complete with respect to IMALL. If $\underline{A} \vdash G$ is provable in IMALL then $\{\underline{A}\}\text{-}\{\} \vdash G$ is provable in IMALL*.

The proof is given in chapter 4.

Lemma 2.2: The system of IMALL* is sound with respect to IMALL. If $\{\underline{A}\}\text{-}\{\} \vdash G$ is provable in IMALL* then $\underline{A} \vdash G$ is provable in IMALL.

The proof is given in chapter 4.

Code for a theorem prover based on IMALL* is given below:

```
prove_imall1(A) :- prove_imall1(A, []).

prove_imall1(A ⊢ G, U) :-
      remove(G, A, U).

prove_imall1(U ⊢ '1', U).

prove_imall1(A ⊢ 'T', U) :-
      (var(U) -> generate_submulti_set(A, U)
      ; submulti_set(A, U)).

prove_imall1(A ⊢ _, U) :-
      remove('0', A, NA),
      (var(U) -> generate_submulti_set(NA, U)
      ; submulti_set(NA, U)).

prove_imall1(A ⊢ C&D, U) :- !,
      prove_imall1(A ⊢ C, U),
      prove_imall1(A ⊢ D, U), !.

prove_imall1(A ⊢ C⊸D, U) :- !,
      prove_imall1([C | A] ⊢ D, U),
      submulti_set(A, U), !.

prove_imall1(A ⊢ G, U) :-
      remove('1', A, NA), !,
      prove_imall1(NA ⊢ G, U), !.

prove_imall1(A ⊢ G, U) :-
      remove(C⊗D, A, NA), !,
      prove_imall1([C, D | NA] ⊢ G, U),
      submulti_set(NA, U), !.

prove_imall1(A ⊢ G, U) :-
      remove(C⊕D, A, NA), !,
      prove_imall1([C | NA] ⊢ G, U),
      prove_imall1([D | NA] ⊢ G, U),
      submulti_set(NA, U), !.

prove_imall1(A ⊢ G, U) :-
      remove(C&D, A, NA),
      (prove([C | NA] ⊢ G, U)
      ;prove([D | NA] ⊢ G, U)
      ),
      submulti_set(NA, U).

prove_imall1(A ⊢ C⊕D, U) :-
      ( prove_imall1(A ⊢ C, U)
      ; prove_imall1(A ⊢ D, U)
      ).

prove_imall1(A ⊢ C⊗D, U) :-
      prove_imall1(A ⊢ C, U1),
      prove_imall1(U1 ⊢ D, U).
```

```
prove_imall1(A ⊢ G, U) :-
      remove(C⊸D, A, NA),
      prove_imall1(NA ⊢ C, U1),
      prove_imall1([D | U1] ⊢ G, U),
      submulti_set(NA, U).

submulti_set(_, []).
submulti_set(A, [F | R]) :-
      remove(F, A, RA),
      submulti_set(RA, R).

generate_submulti_set([], []).
generate_submulti_set([F | R], [F | R1] ) :-
      generate_submulti_set(R, R1).
generate_submulti_set([F | R], R1) :-
      generate_submulti_set(R, R1).
```

This theorem prover, when tested on a range of problems, on average terminated in half the time of the previous one.

Improving efficiency:

The efficiency of the theorem prover can be increase, by modifying the implementation of the axioms. Also goal formula is passed as first argument of prove_imall2 to make use of Prologs' first argument optimization:

```
prove_imall2(A ⊢ G) :-
      check_and_sort_and_prove_imall2(A, [], G, []).

check_and_prove_imall2(G, A, U) :-
      remove(G, A, U).

check_and_prove_imall2('1', A, A).

check_and_prove_imall2(T, A, U) :-
      (var(U) -> generate_submulti_set(A, U)
      ; submulti_set(A, U)).

check_and_prove_imall2(G, A, U) :-
      prove_imall2(G, A, U).

prove_imall2(G, A, U) :-
      remove(F, A, NA), !,
      ( prove_imall2(G, NA, U)
      ; sort_and_prove_imall2([F], NA, G, U)
      ), !.

prove_imall2(C&D, A, U) :- !,
      check_and_prove_imall2(C, A, U),
      check_and_prove_imall2(D, A, U).

prove_imall2(C⊸D, A, U) :- !,
      check_and_sort_and_prove_imall2([C], A, D, U),
      submulti_set(A, U), !.

prove_imall2(G, A, U) :-
      remove('1', A, NA), !,
      prove_imall2(NA, G, U),
      submulti_set(NA, U), !.

prove_imall2(G, A, U) :-
```

```
        remove(C⊗D, A, NA),
        sort_and_prove_imall2([C, D], NA, G, U),
        submulti_set(NA, U).

prove_imall2(G, A, U) :-
        remove(C⊕D, A, NA), !,
        sort_and_prove_imall2([C], NA, G, U),
        sort_and_prove_imall2([D], NA, G, U),
        submulti_set(NA, U).

prove_imall2(G, A, U) :-
        remove(C&D, A, NA), !,
        (sort_and_prove_imall2([C], NA, G, U)
        ;sort_and_prove_imall2([D], NA, G, U)
        ),
        submulti_set(NA, U).

prove_imall2(C⊕D, A, U) :-
        ( check_and_prove_imall2(C, A, U)
        ; check_and_prove_imall2(D, A, U)).

prove_imall2(C⊗D, A, U) :-
        check_and_prove_imall2(C, A, U1),
        check_and_prove_imall2(D, U1, U).

prove_imall2(G, A, U) :-
        remove(C⊸D, A, NA),
        check_and_prove_imall2(C, NA, U1),
        check_and_prove_imall2(G, [D | U1], U),
        submulti_set(NA, U).

sort_and_prove_imall2(F, A, G, U) :-
        remove('0', F, NF),
        (var(U) -> generate_submulti_set(NF, U)
        ; submulti_set(NF, U)).
sort_and_prove_imall2(F, A, G, U) :-
        remove(G, F, NF),
        append(A, NF, U).
sort_and_prove_imall2(F, A, G, U) :-
        append(F, A, NA),
        prove_imall2(G, NA, U).

check_and_sort_and_prove_imall2(F, A, G, U) :-
        remove('0', F, NF),
        (var(U) -> generate_submulti_set(NF, U)
        ; submulti_set(NF, U)).
check_and_sort_and_prove_imall2(F, A, G, U) :-
        append(F, A, NA),
        check_and_prove_imall2(G, NA, U).
```

sort_and_prove_imall2 is called when subproblem is created which is similar to the parent problem but with some additional formulae on the left. These new formulae are checked to see if any one of them is identical to the goal formula or 0. None the 'old' formulae will be identical to the goal formula or 0, otherwise the original problem would be trivial.

check_and_prove_imall2 is called when subproblem is created which is similar to the parent problem but with a new goal formula. The new goal is checked against all formulae on the left to check if the problem is axiomatic. The goal is also checked to see if it is ⊤ or 1, in which case the problem is also axiomatic.

sort_check_and_prove_imall2 is used when the tactic for ⊸ has been applied, as this creates a subproblem with new assumptions and a new goal. This checks for the applicability of all axioms. This predicate is also called when starting the proof.

An attempt was made to store formulae in a data structure giving easy access to the various types of formula. However this proved to be unsuccessful due the increased computation involved in generating a context which is a submulti-set of another.

Efficiency can be further improved by applying the following equivalences (Justified in lemmas 1.19 to 1.46) to all the formulae in a problem, before a call to prove_imall2:

⊤ & D	=	D	0 ⊗ D	=	0
D & ⊤	=	D	D ⊗ 0	=	0
⊤ ⊕ D	=	⊤	0 & D	=	0
D ⊕ ⊤	=	⊤	D & 0	=	0
0 ⊕ D	=	D	D ⊕ 0	=	D
D ⊸ ⊤	=	⊤	0 ⊸ D	=	⊤

This reduces the number of ⊤'s and 0's appearing in the problem. These are the main source of inefficiency in the above theorem prover.

<u>Prolog implementation:</u>

```
reduce_problem(A ⊢ G, NA ⊢ NG) :-
      map(reduce, A, NA),
      reduce(G, NG).

reduce('⊤' & D, D).
reduce(D & '⊤', D).
reduce('⊤' ⊕ D, '⊤').
reduce('0' ⊕ D, D).
reduce(D ⊕ '⊤', '⊤').
reduce(D ⊸ '⊤', '⊤').

reduce(C&D, E) :-
      reduce(C, NC),
      reduce(D, ND),
      reduce2(NC & ND, E).
reduce(C⊕D, E) :-
      reduce(C, NC),
      reduce(D, ND),
      reduce2(NC ⊕ ND, E).
reduce(C⊸D, E) :-
      reduce(C, NC),
      reduce(D, ND),
      reduce2(NC ⊸ ND, E).
reduce(D, D).
```

```
reduce('0' ⊗ D, '0').
reduce(D ⊗ '0', '0').
reduce('0' & D, '0').
reduce(D & '0', '0').
reduce(D ⊕ '0', D).
reduce('0' ⊸ D, '⊤').

reduce(C⊗D, E) :-
      reduce(C, NC),
      reduce(D, ND),
      reduce2(NC ⊗ ND, E).
reduce(C⅋D, E) :-
      reduce(C, NC),
      reduce(D, ND),
      reduce2(NC ⅋ ND, E).
```

```
reduce2('T' & D, D).
reduce2(D & 'T', D).
reduce2('T' ⊕ D, 'T').
reduce2('0' ⊕ D, D).
reduce2(D ⊕ 'T', 'T').
reduce2(D ⊸ 'T', 'T').
reduce2(D, D).
```

```
reduce2('0' ⊗ D, '0').
reduce2(D ⊗ '0', '0').
reduce2('0' & D, '0').
reduce2(D & '0', '0').
reduce2(D ⊕ '0', D).
reduce2('0' ⊸ D, 'T').
```

3.2 MALL Theorem Prover

We define MALL* that contains all the rules of IMALL and the following additional rule:

$$\frac{}{\{a, a^{\perp}, U\}\text{-}\{U\} \vdash \bot}\ I^*$$

If ⊢ $\underline{A}$ is provable in MALL then $\underline{A}^{\perp} \vdash \bot$ is provable in MALL*, given that negated atoms are considered to be atoms. A MALL formula of the form C ⅋ D is considered to be equivalent to the IMALL formula $(C^{\perp} \multimap D)\&(D^{\perp} \multimap C)$.

This system is complete and sound with respect to MALL (Lemmas 3.1 and 3.2). It is implemented by adding the following clause to the MALL theorem prover shown above:

```
prove_imall2(G, A, U) :-
      remove_pair(A, U).

remove_pair([F | R], U) :-
      negate(F, NF),
      remove(NF, R, U), !.
remove_pair([F | R], [F | U]) :-
      remove_pair(R, U).
```

This does not affect the soundness and completeness of the IMALL theorem prover as linear negation cannot occur in an IMALL formula and thus the above clauses cannot succeed.

The formulae in $\underline{A}$ must be negated and transformed so that only atoms are negated:

```
prove_mall(⊢ A) :-
      map(negate, A, A1)
      map(transform_all, A1, NA),
      prove_imall3(NA ⊢ ⊥).

transform_all((A ⊗ B)⊥, C) :- transform_all(A⊥ ⅋ B⊥, C),!.
transform_all((A & B) ⊥ , C) :- transform_all(A⊥ ⊕ B⊥, C),!.
transform_all((A ⊕ B) ⊥ , C) :- transform_all(A⊥ & B⊥, C),!.
transform_all((A ⅋ B) ⊥ , C) :- transform_all(A⊥ ⊗ B⊥, C),!.
transform_all((A ⊸ B) ⊥ , C) :- transform_all(A ⊗ B⊥, C),!.
transform_all(A ⅋ B, C) :-       % ⅋ not a connective of imall
      transform_all((A⊥ ⊸ B)&(B⊥ ⊸ A), C), !.
transform_all(Op(A, B) , Op(NA, NB)) :- !,
      transform_all(A, NA),
      transform_all(B, NB).

transform_all( '0'⊥, 'T').
transform_all( 'T'⊥, '0').
transform_all( '1'⊥, '⊥').
transform_all( '⊥'⊥, '1').
transform_all((A⊥)⊥, C) :- transform_all(A, C).
transform_all(A ⊥ , NA ⊥) :- !,
      transform_all(A, NA).

transform_all( A, A) :- atomic(A).
```

3.3 ILL Theorem Prover

Full ILL with modal operators is undecidable [10]. This can be attributed to the tactic for C!, which allows duplication of formulae. To prevent the possibility of non-termination of the theorem prover, an upper bound is put on the number of times any one banged formula may be duplicated. This is implemented by attaching a number to all queried formulae, representing the maximum number of times that it may be duplicated. Each time that the tactic for C! is applied to duplicate a formula its associated number is decreased by 1, once it reaches 0 that tactic may no longer be used to duplicate the formula.

We thus define the system ILL* that contains all the rules for IMALL* as well as the following:

The tactic for C! is invertible (lemma 25, chapter 5):

$$\frac{\{!C(0),\ !C(N-1),\ \underline{A}\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{!C(N),\ \underline{A}\}\text{-}\{\underline{U}\} \vdash G}\ C! \qquad (N \geq 0)$$

The tactic for !R is only applicable when all formulae on the left are banged. Thus, when modifying this tactic to deal with unused contexts, we immediately know that the unbanged formulae on the left must be unused. This tactic is invertible (lemma 26):

$$\frac{\{!\underline{A1}\}\text{-}\ \{\underline{U}\} \vdash G}{\{!\underline{A1},\ \underline{A2}\}\text{-}\{\underline{U},\ \underline{A2}\} \vdash\ !G}\ !R$$

(Where $\underline{A2}$ is a context in which no formula is banged and $!\underline{A1}$ is a context in which all formulae are banged).

$$\frac{\{\underline{A},\ C\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{\underline{A},\ !C\}\text{-}\{\underline{U}\} \vdash G}\ D!$$

For efficiency reasons, no code for the tactic W! is included. Banged formulae on the left that are returned as unused are simply ignored. This avoids the necessity for the theorem prover to backtrack to apply W! to formulae which need to be thinned.

Lemma 5.1: The system ILL* obtained by adding the above rules to the rules of IMALL* is sound with respect to ILL.

The proof is given in chapter 4.

The system is of course not complete with respect to ILL, but remains complete with respect to IMALL.

Implementation:

```
prove_ill(A ⊢ G, MaxDepth) :-
      check_and_sort_and_prove_ill(A, [], G, U, MaxDepth),
      all_banged(U).

all_banged(U) :- map(banged_formula, U).
banged_formula( ! _).

check_and_prove_ill(G, A, U, MaxDepth) :-
      remove(G, A, U).

check_and_prove_ill('1', A, A, MaxDepth).

check_and_prove_ill(T, A, U, MaxDepth) :-
      (var(U) -> generate_submulti_set(A, U)
      ; submulti_set(A, U)).

check_and_prove_ill(G, A, U, MaxDepth) :-
      prove_ill(G, A, U, MaxDepth).

prove_ill(C&D, A, U, MaxDepth) :- !,
      check_and_prove_ill(C, A, U, MaxDepth),
      check_and_prove_ill(D, A, U, MaxDepth).

prove_ill(C-oD, A, U, MaxDepth) :- !,
      check_and_sort_and_prove_ill([C], A, D, U, MaxDepth),
      submulti_set(A, U).

prove_ill(!G, A, U, MaxDepth) :- !,
      get_banged_context(A, Banged, UnBanged),
      check_and_prove_ill(G, Banged, U1, MaxDepth),
      append(A, UnBanged, U).

prove_ill(G, A, U, MaxDepth) :-
      remove('1', A, NA), !,
      prove_ill(NA, G, U, MaxDepth),
      submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
      remove(C⊗D, A, NA),
      sort_and_prove_ill([C, D], NA, G, U, MaxDepth),
      submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
      remove(C⊕D, A, NA), !,
      sort_and_prove_ill([C], NA, G, U, MaxDepth),
      sort_and_prove_ill([D], NA, G, U, MaxDepth),
      submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
      remove_banged(!C, _, A, NA, MaxDepth),
      sort_and_prove_ill([C], NA, G, U, MaxDepth),
      submulti_set(NA, U).
```

```
prove_ill(G, A, U, MaxDepth) :-
      remove(C&D, A, NA),
      (sort_and_prove_ill([C], NA, G, U, MaxDepth)
      ;sort_and_prove_ill([D], NA, G, U, MaxDepth)
      ),
      submulti_set(NA, U).

prove_ill(C⊕D, A, U, MaxDepth) :-
      ( check_and_prove_ill(C, A, U, MaxDepth)
      ; check_and_prove_ill(D, A, U, MaxDepth)).

prove_ill(C⊗D, A, U, MaxDepth) :-
      check_and_prove_ill(C, A, U1, MaxDepth),
      check_and_prove_ill(D, U1, U, MaxDepth).

prove_ill(G, A, U, MaxDepth) :-
      remove(C⊸D, A, NA),
      check_and_prove_ill(C, NA, U1, MaxDepth),
      check_and_prove_ill(G, [D | U1], U, MaxDepth),
      submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
      remove_banged(!C, D, A, NA, MaxDepth),
      D ≥ 0, !,
      D1 is D - 1,
      sort_and_prove_ill([!(C, 0), !(C, D1)], NA, G, U, MaxDepth),
      submulti_set(NA, U).

sort_and_prove_ill(F, A, G, U, MaxDepth) :-
      remove('0', F, NF),
      (var(U) -> generate_submulti_set(NF, U)
      ; submulti_set(NF, U)).

sort_and_prove_ill(F, A, G, U, MaxDepth) :-
      remove(G, F, NF),
      append(A, NF, U).
sort_and_prove_ill(F, A, G, U, MaxDepth) :-
      append(F, A, NA),
      prove_ill(G, NA, U, MaxDepth).

check_and_sort_and_prove_ill(F, A, G, U, MaxDepth) :-
      remove('0', F, NF),
      (var(U) -> generate_submulti_set(NF, U)
      ; submulti_set(NF, U)).
check_and_sort_and_prove_ill(F, A, G, U, MaxDepth) :-
      append(F, A, NA),
      check_and_prove_ill(G, NA, U, MaxDepth).

remove_banged(!F, MD, A, U, MD) :-
      remove(!F, A, U).
remove_banged(!F, D, A, U, _) :-
      remove(!(F, D), A, U).

get_banged_context([], [], []).
get_banged_context([!F | R], [!F | B], UB) :- !,
      get_banged_context(R, B, UB).
get_banged_context([F | R], B, [F | UB]) :- !,
      get_banged_context(R, B, UB).
```

3.4 An Alternative IMALL Theorem Prover

In the IMALL theorem prover described earlier, the tactics for 0 and $\top$ pose efficiency problems as they are required to repeatedly return as unused a subset of the input context until a suitable one is found. Determining all subsets of a list of formulae takes time exponential in the number of formulae.

To overcome, instead of returning a subset of the input context, the tactics of $\top$ and 0 will return all the formulae in the input context, but mark each one as thinnable, indicating that it need not be consumed:

$$\frac{}{\{\underline{A}\}\text{-}\{\mathrm{th}{:}\underline{A}\} \vdash \top}\ \top^{*} \qquad\qquad \frac{}{\{\underline{A}, 0\}\text{-}\{\mathrm{th}{:}\underline{A}\} \vdash G}\ 0^{*}$$

Where th:$\underline{A}$ is a context that contains the same formulae as $\underline{A}$, except that all formulae are marked as being thinnable.

All formulae are initially marked as unthinnable by being preceded by the marker 'un:'. Thus the ILL sequent $\underline{A} \vdash G$ translates to the ILL* sequent $\{\mathrm{un}{:}\underline{A}\}\text{-}\{\} \vdash G$.

The fact that a thinnable formula may or may not be used is expressed by the following two tactics:

$$\frac{\{\mathrm{un}{:}C, \underline{A}\}\text{-}\{\underline{U}\} \vdash G}{\{\mathrm{th}{:}C, \underline{A}\}\text{-}\{\underline{U}\} \vdash G}\ \text{De-thin}$$

$$\frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash G}{\{\mathrm{th}{:}C, \underline{A}\}\text{-}\{\underline{U}\} \vdash G}\ \text{Thin}$$

Tactics may now only be applied to unthinnable formulae, which are marked as such by being preceded by 'un:'. Thinnable formulae have to be De-thinned in order to have tactics applied to them.

The translation from a ILL* sequent to an ILL sequent is shown below:

$$\begin{array}{lclcl}
\{\underline{A}\}\text{-}\{\} \vdash G & = & \underline{A} \vdash G & & \\
\{\mathrm{un}{:}C, \underline{A}\}\text{-}\{\mathrm{un}{:}C, \underline{U}\} \vdash G & = & \{\underline{A}\}\text{-}\{\underline{U}\} \vdash G & & \\
\{\mathrm{th}{:}C, \underline{A}\}\text{-}\{\mathrm{th}{:}C, \underline{U}\} \vdash G & = & \{\underline{A}\}\text{-}\{\underline{U}\} \vdash G & & \\
\{\mathrm{un}{:}C, \underline{A}\}\text{-}\{\mathrm{th}{:}C, \underline{U}\} \vdash G & = & \{\underline{A}\}\text{-}\{\underline{U}\} \vdash G & \text{or} & \{\mathrm{un}{:}C, \underline{A}\}\text{-}\{\underline{U}\}
\end{array}$$

Thus $\{\mathrm{un}{:}\underline{A}\}\text{-}\{\mathrm{th}{:}\underline{A}\}$ denotes any subset of A, as desired.

An examination of the tactics will show that a sequent of the form $\{\mathrm{th}{:}C, \underline{A}\}\text{-}\{\mathrm{un}{:}C, \underline{U}\} \vdash G$ cannot occur (Unless un:$\underline{C}$ occurs elsewhere in $\underline{A}$).

IMALL* tactics:

$$\frac{\{\underline{A}\}\text{-}\{\underline{U1}\} \vdash C \qquad \{\underline{U1}\}\text{-}\{\underline{U}\} \vdash D}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \otimes D}\ \otimes R^{*} \qquad \frac{\{\underline{A}\}\text{-}\{\underline{U1}\} \vdash C \qquad \{\underline{U1}, D\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{\mathrm{un}{:}C \multimap D, \underline{A}\}\text{-}\{\underline{U}\} \vdash G}\ {\multimap}L^{*}$$

$$\frac{\{\underline{A}, \mathrm{un}{:}C\}\text{-}\{\underline{U}\} \vdash D \quad \underline{U} \subseteq \underline{A}}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \multimap D}\ {\multimap}R^{*} \qquad \frac{\{\underline{A}, \mathrm{un}{:}C\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{\underline{A}, \mathrm{un}{:}C \& D\}\text{-}\{\underline{U}\} \vdash G}\ \&L1^{*}$$

$$\frac{\{\underline{A}, \mathrm{un}{:}D\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{\underline{A}, \mathrm{un}{:}C \& D\}\text{-}\{\underline{U}\} \vdash G}\ \&L2^{*} \qquad \frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \quad \{\underline{A}\}\text{-}\{\underline{U}\} \vdash D}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \& D}\ \&R^{*}$$

$$\frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \oplus D}\ \oplus R1^* \qquad \frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash D}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \oplus D}\ \oplus R2^*$$

$$\frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash G}{\{\underline{A}, \text{un:}1\ \}\text{-}\{\underline{U}\} \vdash G}\ 1L^* \qquad \frac{\{\underline{A}, \text{un:}C\}\text{-}\{\underline{U}\} \vdash G \quad \{\underline{A}, \text{un:}D\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{\underline{A}, \text{un:}C \oplus D\}\text{-}\{\underline{U}\} \vdash G}\ \oplus L^*$$

$$\frac{\{\underline{A}, \text{un:}C, \text{un:}D\}\text{-}\{\underline{U}\} \vdash G}{\{\underline{A}, \text{un:}C \otimes D\}\text{-}\{\underline{U}\} \vdash G}\ \otimes L^*$$

$$\frac{}{\{\text{un:}G, \underline{A}\}\text{-}\{\underline{A}\} \vdash G}\ I^*$$

$$\frac{}{\{\underline{A}\}\text{-}\{\underline{A}\} \vdash 1}\ 1^*$$

$$\frac{}{\{\underline{A}, \text{un:}0\}\text{-}\{\text{th:}\underline{A}\} \vdash G}\ 0^*$$

$$\frac{}{\{\underline{A}\}\text{-}\{\text{th:}\underline{A}\} \vdash T}\ T^*$$

$$\frac{\{\text{un:}C, \underline{A}\}\text{-}\{\underline{U}\} \vdash G}{\{\text{th:}C, \underline{A}\}\text{-}\{\underline{U}\} \vdash G}\ \text{De-thin} \qquad \frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash G}{\{\text{th:}C, \underline{A}\}\text{-}\{\underline{U}\} \vdash G}\ \text{Thin}$$

The following tactics are also valid and are added to the system for efficiency:

$$\frac{}{\{\text{th:}G, \underline{A}\}\text{-}\{\underline{A}\} \vdash G}\ I^*$$

$$\frac{}{\{\underline{A}, \text{th:}0\}\text{-}\{\text{th:}\underline{A}\} \vdash G}\ 0^*$$

$$\frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash G}{\{\underline{A}\}\text{-}\{\underline{U}\}, \text{th:}1 \vdash G}\ 1L^*$$

I believe this system to be sound and complete with respect to IMALL but have been unable to derive a proof of this.

The implementation of this system is shown below:

```
prove_imall3(A ⊢ G) :-
      mark_unthinnable(A, NA),
      sort_check_and_prove_imall3(NA, [] ⊢ G, U),
      all_thinnable(U). % Thinnable formulae may be ignored

check_and_prove_imall3(A ⊢ G, U) :-
      remove(_:G, A, U).

check_and_prove_imall3(A ⊢ '1', A).

check_and_prove_imall3(A ⊢ T, U) :-
      (var(U) -> mark_thinnable(A, U)
      ;true).

check_and_prove_imall3(A ⊢ G, U) :-
      prove_imall3(A ⊢ G, U).
```

```
prove_imall3(A ⊢ G, U) :-
      remove(th:F, A, NA), !,
      ( prove_imall3(NA ⊢ G, U)
      ; sort_and_prove_imall3([un:F], NA ⊢ G, U)
      ).

prove_imall3(A ⊢ C&D, U) :- !,
      check_and_prove_imall3(A ⊢ C, U),
      check_and_prove_imall3(A ⊢ D, U).

prove_imall3(A ⊢ C⊸D, U) :- !,
      sort_check_and_prove_imall3([un:C], A ⊢ D, U),
      subset(A, U).

prove_imall3(A ⊢ G, U) :-
      remove(_:'1', A, NA), !,
      prove_imall3(NA ⊢ G, U),
      subset(NA, U).

prove_imall3(A ⊢ G, U) :-
      remove(un:C⊗D, A, NA), !,
      sort_and_prove_imall3([un:C, un:D], NA ⊢ G, U),
      subset(NA, U).

prove_imall3(A ⊢ G, U) :-
      remove(un:C⊕D, A, NA), !,
      sort_and_prove_imall3([un:C], NA ⊢ G, U),
      sort_and_prove_imall3([un:D], NA ⊢ G, U),
      subset(NA, U).

prove_imall3(A ⊢ G, U) :-
      remove(un:C&D, A, NA), !,
      (sort_and_prove_imall3([un:C], NA ⊢ G, U)
      ;sort_and_prove_imall3([un:D], NA ⊢ G, U)
      ),
      subset(NA, U).

prove_imall3(A ⊢ C⊕D, U) :-
      ( check_and_prove_imall3(A ⊢ C, U)
      ; check_and_prove_imall3(A ⊢ D, U)).

prove_imall3(A ⊢ C⊗D, U) :-
      check_and_prove_imall3(A ⊢ C, U1),
      check_and_prove_imall3(U1 ⊢ D, U).

prove_imall3(A ⊢ G, U) :-
      remove(un:C⊸D, A, NA),
      check_and_prove_imall3(NA ⊢ C, U1),
      check_and_prove_imall3([un:D | U1] ⊢ G, U),
      subset(NA, U).

sort_and_prove_imall3(F, A ⊢ G, U) :-
      remove('0', F, NF),
      (var(U) -> mark_thinnable(NF, U)
      ;true).
```

```
sort_and_prove_imall3(F, A ⊢ G, U) :-
      remove(_:G, F, NF),
      append(A, NF, U).
sort_and_prove_imall3(F, A ⊢ G, U) :-
      append(F, A, NA),
      prove_imall3(NA ⊢ G, U).

sort_check_and_prove_imall3(F, A ⊢ G, U) :-
      remove(_:'0', F, NF),
      (var(U) -> mark_thinnable(NF, U)
      ;true).
sort_check_and_prove_imall3(F, A ⊢ G, U) :-
      append(F, A, NA),
      check_and_prove_imall3(NA ⊢ G, U).

mark_unthinnable(A, B) :- map(mark_unthinnable_formula, A, B).
mark_unthinnable_formula(A, un:A).

mark_thinnable(A, B) :- map(mark_thinnable_formula, A, B).
mark_thinnable_formula(_:A, un:A).

all_thinnable(U) :- map(=(th:_), U).
```

The predicate subset is modified to allow for thinnable formulae according to the definition given above.

4 LEMMAS

4.1 ILL lemmas

Lemma 1.1: A sequent of the form E ⊢ E where E is a formula is always provable.

Proof: We proceed by induction on size of E. By the I, axiom the hypothesis is true if E has size 1 (ie is an atom).

Suppose E has the form C⊕D:

$$\dfrac{\dfrac{C \vdash C}{C \vdash C\oplus D}\oplus R1 \qquad \dfrac{D \vdash D}{D \vdash C\oplus D}\oplus R2}{C\oplus D \vdash C\oplus D}\oplus L$$

By induction hypothesis C ⊢ C and D ⊢ D are provable.

Suppose E has the form C⊗D:

$$\dfrac{\dfrac{C \vdash C \quad D \vdash D}{C,\, D \vdash C\otimes D}\otimes R}{C\otimes D \vdash C\otimes D}\otimes L$$

By induction hypothesis C ⊢ C and D ⊢ D are provable.

Suppose E has the form C&D:

$$\dfrac{\dfrac{C \vdash C}{C\&D \vdash C}\&L1 \qquad \dfrac{D \vdash D}{C\&D \vdash D}\&L2}{C\&D \vdash C\&D}\&R$$

By induction hypothesis C ⊢ C and D ⊢ D are provable.

Suppose E has the form A⊸B:

$$\dfrac{\dfrac{C \vdash C \quad D \vdash D}{C\multimap D,\, C \vdash D}\multimap L}{C\multimap D \vdash C\multimap D}\multimap R$$

By induction hypothesis C ⊢ C and D ⊢ D are provable.

Suppose E has the form !D ⊢ !D:

$$\dfrac{\dfrac{D \vdash D}{!D \vdash D}\,!L}{!D \vdash\ !D}\,!R$$

By induction hypothesis D ⊢ D is provable.

Thus the hypothesis holds in all cases. ■

Lemma 1.2: The ILL rule of $\otimes$R is not invertible.

Proof: We need to find a provable problem of the form $\underline{A} \vdash C\otimes D$ such that for any partition $\underline{A1}$, $\underline{A2}$ of $\underline{A}$, $\underline{A1} \vdash C$ or $\underline{A2} \vdash D$ is not provable.

$$p\otimes q \vdash p\otimes q \text{ is provable: } \dfrac{\dfrac{p \vdash p \quad q \vdash q}{p, q \vdash p\otimes q}\otimes R}{p\otimes q \vdash p\otimes q}\otimes L$$

but none of $\vdash p$, $\vdash q$, $p\otimes q \vdash p$ or $p\otimes q \vdash q$ is provable. ■

Lemma 1.3: The ILL rule of &L1 is not invertible.

Proof: We need to find a provable problem of the form $C\&D, \underline{A} \vdash G$ such that $C, \underline{A} \vdash G$ is not provable.

$$p\&q \vdash q \text{ is provable: } \dfrac{\dfrac{}{q \vdash q}I}{p\&q \vdash q}\&L2$$

but $p \vdash q$ is not. ■

Lemma 1.4: The ILL rule of &L2 is not invertible.

Proof: Similar to &L1 ■

Lemma 1.5: The ILL rule of $\oplus$R1 is not invertible.

Proof: We need to find a provable problem of the form $\underline{A} \vdash C\oplus D$ such that $\underline{A} \vdash C$ is not provable.

$$q \vdash p\oplus q \text{ is provable: } \dfrac{\dfrac{}{q \vdash q}I}{q \vdash p\oplus q}\oplus R2$$

but $q \vdash p$ is not. ■

Lemma 1.6: The ILL rule of $\oplus$R2 is not invertible.

Proof: Similar to $\oplus$L2 ■

Lemma 1.7: The ILL rule of $\multimap$L is not invertible.

Proof: We need to find a provable problem of the form $\underline{A}, C\multimap D \vdash G$ such that for any partition $\underline{A1}$, $\underline{A2}$ of $\underline{A}$, $\underline{A1} \vdash C$ or $\underline{A2}, D \vdash G$ is not provable.

$p\multimap q \vdash p\multimap q$ is provable (lemma 1.1).

but $\vdash p$ and $q \vdash p\multimap q$ are not. ■

Lemma 1.8: The ILL rule of !W is not invertible.

Proof: We need to find a provable problem of the form $!C, \underline{A} \vdash G$ such that $\underline{A} \vdash G$ is not provable.

$$!p \vdash p \text{ is provable: } \dfrac{\dfrac{}{p \vdash p}I}{!p \vdash p}!D$$

but $\vdash p$ is not. ■

Lemma 1.9: The ILL rule of !D is not invertible.

Proof: We need to find a provable problem of the form $!D, \underline{A} \vdash G$ such that $D, \underline{A} \vdash G$ is not provable.

$!p \vdash\ !p$ is provable (lemma 1.1)

but $p \vdash\ !p$ is not. ■

Lemma 1.10: The ILL rule of !C is invertible.

Proof: We need to show that if $!D, \underline{A} \vdash G$ is provable then $!D,\ !D, \underline{A} \vdash G$ is provable. We construct the proofs as shown:

$$\frac{!D, \underline{A} \vdash G}{!D,\ !D, \underline{A} \vdash G}\ !W \quad ■$$

Lemma 1.11: The ILL rule of &R is invertible.

Proof: We need to show that if $\underline{A} \vdash C\&D$ is provable then $\underline{A} \vdash C$ and $\underline{A} \vdash D$ are provable. We construct the proofs as shown:

$$\frac{\underline{A} \vdash C\&D \quad \dfrac{C \vdash C}{C\&D \vdash C}\ \&L1}{\underline{A} \vdash C}\ \mathrm{Cut}$$

$$\frac{\underline{A} \vdash C\&D \quad \dfrac{D \vdash D}{C\&D \vdash D}\ \&L1}{\underline{A} \vdash D}\ \mathrm{Cut}$$

By lemma 1.1 $C \vdash C$ and $D \vdash D$ are provable ■

Lemma 1.12: The ILL rule of ⊗L is invertible:

Proof: We need to show that if $C \otimes D, \underline{A} \vdash G$ is provable then C, D, A ⊢ G
We construct the proof as sh

Lemma 1.2: The ILL rule of ⊗R is not invertible.

Proof: We need to find a provable problem of the form $\underline{A} \vdash C \otimes D$ such that for any partition $\underline{A1}, \underline{A2}$ of $\underline{A}$, $\underline{A1} \vdash C$ or $\underline{A2} \vdash D$ is not provable.

$p \otimes q \vdash p \otimes q$ is provable: $$\frac{\dfrac{p \vdash p \quad q \vdash q}{p, q \vdash p \otimes q}\ \otimes R}{p \otimes q \vdash p \otimes q}\ \otimes L$$

but none of $\vdash p$, $\vdash q$, $p \otimes q \vdash p$ or $p \otimes q \vdash q$ is provable. ■

Lemma 1.3: The ILL rule of &L1 is not invertible.

Proof: We need to find a provable problem of the form $C\&D, \underline{A} \vdash G$ such that $C, \underline{A} \vdash G$ is not provable.

$p\&q \vdash q$ is provable: $$\frac{\dfrac{}{q \vdash q}\ I}{p\&q \vdash q}\ \&L2$$

but $p \vdash q$ is not. ■

Lemma 1.14: The ILL rule of ⊕L is invertible:

Proof: We need to show that if $C\oplus D, \underline{A} \vdash G$ is provable then $C, \underline{A} \vdash G$ and $D, \underline{A} \vdash G$ are provable. We construct the proofs as shown:

$$\dfrac{\dfrac{C \vdash C}{C \vdash C\oplus D}\oplus R1 \qquad C\oplus D, \underline{A} \vdash G}{C, \underline{A} \vdash G}\,\text{Cut}$$

$$\dfrac{\dfrac{D \vdash D}{D \vdash C\oplus D}\oplus R1 \qquad C\oplus D, \underline{A} \vdash G}{D, \underline{A} \vdash G}\,\text{Cut}$$

By lemma 1.1 $C \vdash C$ and $D \vdash D$ are provable ■

Lemma 1.15: The rule of !R is invertible.

Proof: Have to show that if $!\underline{A} \vdash\ !G$ is provable, then $!\underline{A} \vdash G$ is provable. We construct the proof as shown:

$$\dfrac{!\underline{A} \vdash\ !G \qquad \dfrac{G \vdash G}{!G \vdash G}\,!L2}{!\underline{A} \vdash G}\,\text{Cut}$$

By lemma 1.1 $G \vdash G$ is provable ■

Lemma 1.16: The rule of ⊸R is invertible.

Proof: We need to show that if $\underline{A} \vdash C\multimap D$ is provable then $\underline{A}, C \vdash D$ is provable. We construct the proof as shown:

$$\dfrac{\underline{A} \vdash C\multimap D \qquad \dfrac{C \vdash C \quad D \vdash D}{C\multimap D \vdash D}\multimap L}{\underline{A}, C \vdash D}\,\text{Cut}$$

By lemma 1.1 $C \vdash C$ and $D \vdash D$ are provable ■

Lemma 1.17: If $\underline{A}, \underline{B} \vdash G$, where $\underline{A}$ contains formulae A1, A2, ..., An is provable then $A1\otimes A2\otimes \ldots \otimes An, \underline{B} \vdash G$ is provable.

Proof: By induction on n.

Proof for n = 2:

$$\dfrac{A1, A2, \underline{B} \vdash G}{A1\otimes A2, \underline{B} \vdash G}\otimes$$

Proof for all n:

$$\dfrac{A1\otimes A2\otimes \ldots An\text{-}1, An, \underline{B} \vdash G}{A1\otimes A2\otimes \ldots An\text{-}1\otimes An, \underline{B} \vdash G}\otimes$$

By induction hypothesis $A1\otimes A2\otimes \ldots An\text{-}1, \underline{B}, An \vdash G$ is provable if $A1, A2, \ldots An\text{-}1, \underline{B}, An \vdash G$ is provable. Thus $A1\otimes A2\otimes \ldots An\text{-}1\otimes An, \underline{B} \vdash G$ is provable if $A1, A2, \ldots An\text{-}1, An, \underline{B} \vdash G$ is provable. ■

Lemma 1.18: If A1⊗A2⊗ ... ⊗An, B̲ ⊢ G is provable then A̲, B̲ ⊢ G , where A̲ contains formulae A1, A2, ..., An is provable.

Proof: By induction on n.

Suppose n = 2:

$$\dfrac{\dfrac{\overline{A1 \vdash A1}\ I \qquad \overline{A2 \vdash A2}\ I}{A1,\ A2 \vdash A1\otimes A2}\ \otimes L \qquad \underline{B},\ A1\otimes A2 \vdash G}{A1,\ A2,\ \underline{B} \vdash G}\ \text{Cut}$$

Proof for any n:

$$\dfrac{\dfrac{\overline{A1 \vdash A1}\ I \qquad A2,\ldots,An \vdash A2\otimes \ldots\otimes An}{A1,\ A2\ \ldots,An \vdash A1\otimes A2\ldots\otimes An}\ \text{⅋} \qquad A1\otimes A2\otimes\ldots\otimes An,\ \underline{B} \vdash G}{A1,\ A2,\ \ldots\ An,\ \underline{B} \vdash G}\ \text{Cut}$$

By induction hypothesis A2,...,An ⊢ A2⊗ ...⊗An will be provable if A2⊗...⊗An ⊢ A2⊗ ...⊗An is provable.
A2⊗...⊗An ⊢ A2⊗ ...⊗An is provable by lemma 1.1. ■

Lemma 1.19: If A, T&D ⊢ G is provable then A, D ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{\dfrac{\overline{\vdash T}\ T \qquad \overline{D \vdash D}\ I}{D \vdash T\&D}\ \&R \qquad A,\ T\&D \vdash G}{A,\ D \vdash G}\ \text{Cut}\quad ■$$

Lemma 1.20: If A, D ⊢ G is provable then A, T&D ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{A,\ D \vdash G \qquad \dfrac{\overline{D \vdash D}\ I}{T\&D \vdash D}\ \&L2}{A,\ T\&D \vdash G}\quad ■$$

Lemma 1.21: If A ⊢ T&D is provable then A ⊢ D is provable.

Proof: We construct the proof as shown:

$$\dfrac{A \vdash T\&D \qquad \dfrac{D \vdash D}{T\&D \vdash D}\ \&L2}{A \vdash D}\ \text{Cut}\quad ■$$

Lemma 1.22: If A ⊢ D is provable then A ⊢ T&D is provable.

Proof: We construct the proof as shown:

$$\dfrac{A \vdash D \qquad \dfrac{\overline{D \vdash T}\ T \qquad \overline{D \vdash D}\ I}{D \vdash T\&D}\ \&R}{A \vdash T\&D}\quad ■$$

Lemma 1.23: If A, 0⊗D ⊢ G is provable then A, 0 ⊢ G is provable.

Proof: We construct the proof as shown:

$$\frac{A, 0\otimes D \vdash G \qquad \dfrac{}{0 \vdash 0\otimes D}\,0}{A, 0 \vdash G}\,\text{Cut} \qquad \blacksquare$$

Lemma 1.24: If A, 0 ⊢ G is provable then A, 0⊗D ⊢ G is provable.

Proof: We construct the proof as shown:

$$\frac{A, 0 \vdash G \qquad \dfrac{\dfrac{}{0, D \vdash 0}\,0}{0\otimes D \vdash 0}\,\otimes L}{A, 0\otimes D \vdash G} \qquad \blacksquare$$

Lemma 1.25: If A ⊢ 0⊗D is provable then A ⊢ 0 is provable.

Proof: We construct the proof as shown:

$$\frac{A \vdash 0\otimes D \qquad \dfrac{\dfrac{}{0, D \vdash 0}\,0}{0\otimes D \vdash 0}\,\otimes L}{A \vdash 0} \qquad \blacksquare$$

Lemma 1.26: If A ⊢ 0 is provable then A ⊢ 0⊗D is provable.

Proof: We construct the proof as shown:

$$\frac{A \vdash 0 \qquad \dfrac{}{0 \vdash 0\otimes D}\,0}{A \vdash 0\otimes D} \qquad \blacksquare$$

Lemma 1.27: If A, T⊕D ⊢ G is provable then A, T ⊢ G is provable.

Proof: We construct the proof as shown:

$$\frac{A, T\oplus D \vdash G \qquad \dfrac{\dfrac{}{T \vdash T}\,I}{T \vdash T\oplus D}\,\oplus R1}{A, T \vdash G}\,\text{Cut} \qquad \blacksquare$$

Lemma 1.28: If A, T ⊢ G is provable then A, T⊕D ⊢ G is provable.

Proof: We construct the proof as shown:

$$\frac{A, T \vdash G \qquad \dfrac{}{T\oplus D \vdash T}\,T}{A, T\oplus D \vdash G} \qquad \blacksquare$$

Lemma 1.29: If A ⊢ T⊕D is provable then A ⊢ T is provable.

Proof: We construct the proof as shown:

$$\frac{A \vdash T\oplus D \qquad \dfrac{}{T\oplus D \vdash T}\,I}{A \vdash T}\,\text{Cut} \qquad \blacksquare$$

Lemma 1.30: If A ⊢ ⊤ is provable then A ⊢ ⊤⊕D is provable.

Proof: We construct the proof as shown:

$$\dfrac{A \vdash \top \qquad \dfrac{\dfrac{}{\top \vdash \top}\,\mathrm{I}}{\top \vdash \top \oplus D}\,\oplus\mathrm{R}1}{A \vdash \top \oplus D} \qquad \blacksquare$$

Lemma 1.31: If A, 0&D ⊢ G is provable then A, 0 ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{A, 0\&D \vdash G \qquad \dfrac{\dfrac{}{0 \vdash 0}\,\mathrm{I}}{0 \vdash 0\&D}\,\&\mathrm{R}1}{A, 0 \vdash G}\,\mathrm{Cut} \qquad \blacksquare$$

Lemma 1.32: If A, 0 ⊢ G is provable then A, 0&D ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{A, 0 \vdash G \qquad \dfrac{\dfrac{}{0 \vdash 0}\,\mathrm{I}}{0\&D \vdash 0}\,\&\mathrm{L}1}{A, 0\&D \vdash G} \qquad \blacksquare$$

Lemma 1.33: If A ⊢ 0&D is provable then A ⊢ 0 is provable.

Proof: We construct the proof as shown:

$$\dfrac{A \vdash 0\&D \qquad \dfrac{\dfrac{}{0 \vdash 0}\,\mathrm{I}}{0\&D \vdash 0}\,\&\mathrm{L}1}{A \vdash 0}\,\mathrm{Cut} \qquad \blacksquare$$

Lemma 1.34: If A ⊢ 0 is provable then A ⊢ 0&D is provable.

Proof: We construct the proof as shown:

$$\dfrac{A \vdash 0 \qquad \dfrac{}{0 \vdash 0\&D}\,0}{A \vdash 0\&D} \qquad \blacksquare$$

Lemma 1.35: If A, 0⊕D ⊢ G is provable then A, D ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{A, 0\oplus D \vdash G \qquad \dfrac{\dfrac{}{D \vdash D}\,\mathrm{I}}{D \vdash 0\oplus D}\,\oplus\mathrm{R}2}{A, D \vdash G}\,\mathrm{Cut} \qquad \blacksquare$$

Lemma 1.36: If A, D ⊢ G is provable then A, 0⊕D ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{A, D \vdash G \qquad \dfrac{\dfrac{}{0 \vdash D}\,0 \qquad \dfrac{}{D \vdash D}\,I}{0 \oplus D \vdash D}\,\oplus L}{A, 0 \oplus D \vdash G}$$

■

Lemma 1.37: If A ⊢ 0⊕D is provable then A ⊢ D is provable.

Proof: We construct the proof as shown:

$$\dfrac{A \vdash 0 \oplus D \qquad \dfrac{\dfrac{}{0 \vdash D}\,0 \qquad \dfrac{}{D \vdash D}\,I}{0 \oplus D \vdash D}\,\oplus L}{A \vdash D}\,\mathrm{Cut}$$

■

Lemma 1.38: If A ⊢ D is provable then A ⊢ 0⊕D is provable.

Proof: We construct the proof as shown:

$$\dfrac{A \vdash D \qquad \dfrac{\dfrac{}{D \vdash D}\,I}{D \vdash 0 \oplus D}\,\oplus R2}{A \vdash 0 \oplus D}$$

■

Lemma 1.39: If A, D⊸⊤ ⊢ G is provable then A, ⊤ ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{A, D \multimap \top \vdash G \qquad \dfrac{\dfrac{}{\top, D \vdash \top}\,\top}{\top \vdash D \multimap \top}\,\multimap R}{A, \top \vdash G}\,\mathrm{Cut}$$

■

Lemma 1.40: If A, ⊤ ⊢ G is provable then A, D⊸⊤ ⊢ G is provable.

Proof: We construct the proof as shown:

$$\dfrac{A, \top \vdash G \qquad \dfrac{\dfrac{}{\top, D \vdash \top}\,\top}{\top \vdash D \multimap \top}}{A, D \multimap \top \vdash G}$$

■

Lemma 1.41: If A ⊢ D⊸⊤ is provable then A ⊢ ⊤ is provable.

Proof: We construct the proof as shown:

$$\dfrac{}{A \vdash \top}\,\top$$

■

Lemma 1.42: If A ⊢ ⊤ is provable then A ⊢ D⊸⊤ is provable.

Proof: We construct the proof as shown:

$$\dfrac{\dfrac{}{A, D \vdash \top}\,\top}{A \vdash D \multimap \top}\,\multimap R$$

■

Lemma 1.43: If $A, 0 \multimap D \vdash G$ is provable then $A, \top \vdash G$ is provable.

Proof: We construct the proof as shown:

$$\frac{A, 0 \multimap D \vdash G \qquad \dfrac{\dfrac{}{\top, 0 \vdash D}\;0}{\top \vdash 0 \multimap D}\;\multimap R}{A, \top \vdash G}\;\text{Cut} \qquad \blacksquare$$

Lemma 1.44: If $A, \top \vdash G$ is provable then $A, 0 \multimap D \vdash G$ is provable.

Proof: We construct the proof as shown:

$$\frac{A, \top \vdash G \qquad \dfrac{}{0 \multimap D \vdash \top}\;\top}{A, 0 \multimap D \vdash G}\;\text{Cut} \qquad \blacksquare$$

Lemma 1.45: If $A \vdash 0 \multimap D$ is provable then $A \vdash \top$ is provable.

Proof: We construct the proof as shown:

$$\frac{}{A \vdash \top}\;\top \qquad \blacksquare$$

Lemma 1.46: If $A \vdash \top$ is provable then $A \vdash 0 \multimap D$ is provable.

Proof: We construct the proof as shown:

$$\frac{A, 0 \vdash D}{A \vdash 0 \multimap D}\;\multimap R \qquad \blacksquare$$

4.2 IMALL* lemmas

Lemma 2.1 If $\underline{A} \vdash G$ is provable in IMALL then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL*, for any $\underline{U}$. In particular if $\underline{A} \vdash G$ is provable in IMALL then $\{\underline{A}\}\text{-}\{\} \vdash G$ is provable in IMALL*.

Proof: We proceed by induction on the height of the IMALL proof:

Suppose that the proof ends in the application of the axiom of I:

$$\frac{}{A \vdash A}\ I$$

then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash A$ is provable in IMALL*

$$\frac{}{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash A}\ I^*$$

Suppose that the proof ends in the application of the axiom of 1:

$$\frac{}{\vdash 1}\ 1$$

then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash 1$ is provable in IMALL*

$$\frac{}{\{\underline{A}, \underline{U}\}\text{-}\{\underline{A}\} \vdash 1}\ 1^*$$

Suppose that the proof ends in the application of the axiom of 0:

$$\frac{}{\underline{A}, 0 \vdash G}\ 0$$

then $\{\underline{A}, 0, \underline{U}\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL*

$$\frac{}{\{\underline{A}, 0, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}\ 0^*\ (\underline{U} \text{ is a subset of } \underline{U}, 0)$$

Suppose that the proof ends in the application of the axiom of ⊤:

$$\frac{}{\underline{A} \vdash T}\ T$$

then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash T$ is provable in IMALL*

$$\frac{}{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash T}\ T^*\ (\underline{U} \text{ is a subset of } \underline{U})$$

Suppose that the proof ends in the application of the rule of ⊗R:

$$\frac{\underline{A1} \vdash C \qquad \underline{A2} \vdash D}{\underline{A1}, \underline{A2} \vdash C \otimes D}\ \otimes R$$

then $\{\underline{A2}, \underline{A2}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C \otimes D$ is provable in IMALL*

$$\frac{\{\underline{A1}, \underline{A2}, \underline{U}\}\text{-}\{\underline{A2}, \underline{U}\} \vdash C \qquad \{\underline{A2}, \underline{U}\}\text{-}\{\underline{U}\} \vdash D}{\{\underline{A1}, \underline{A2}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C\otimes D}\ \otimes R^*$$

By induction hypothesis if $\underline{A1} \vdash C$ and $\underline{A2} \vdash D$ are provable in MALL then $\{\underline{A1}, \underline{A2}, \underline{U}\}$-$\{\underline{A2}, \underline{U}\} \vdash C$ and $\{\underline{A2}, \underline{U}\}$-$\{\underline{U}\} \vdash D$ are provable in IMALL*.

Suppose that the proof ends in the application of the rule of $\multimap$L:

$$\frac{\underline{A1} \vdash C \qquad D, \underline{A2} \vdash G}{C\multimap D, \underline{A1}, \underline{A2} \vdash G}\ \multimap L$$

then $\{C\multimap D, \underline{A1}, \underline{A2}, \underline{U}\}$-$\{\underline{U}\} \vdash G$ is provable in IMALL*

$$\frac{\{\underline{A1}, \underline{A2}, \underline{U}\}\text{-}\{\underline{A2}, \underline{U}\} \vdash C \qquad \{\underline{A2}, \underline{U}\}\text{-}\{\underline{U}\} \vdash G \qquad \overline{\underline{U} \subseteq \underline{U}, \underline{A1}, \underline{A2}}}{\{C\multimap D, \underline{A1}, \underline{A2}, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}\ \multimap L^*$$

By induction hypothesis if $\underline{A1} \vdash C$ and $D, \underline{A2} \vdash G$ are provable in MALL then $\{\underline{A1}, \underline{A2}, \underline{U}\}$-$\{\underline{A2}, \underline{U}\} \vdash C$ and $\{\underline{A2}, \underline{U}\}$-$\{\underline{U}\} \vdash G$ are provable in IMALL*.

Suppose that the proof ends in the application of the rule of $\multimap$R:

$$\frac{\underline{A}, C \vdash D}{\underline{A} \vdash C\multimap D}\ \multimap R$$

then $\{\underline{A}, \underline{U}\}$-$\{\underline{U}\} \vdash C\multimap D$ is provable in IMALL*

$$\frac{\{\underline{A}, C, \underline{U}\}\text{-}\{\underline{U}\} \vdash D \qquad \overline{\underline{U} \subseteq \underline{A}, \underline{U}}}{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C\multimap D}\ \multimap R^*$$

By induction hypothesis if $\underline{A}, C \vdash D$ is provable in MALL then $\{\underline{A}, C, \underline{U}\}$-$\{\underline{U}\} \vdash D$ is provable in IMALL*.

Suppose that the proof ends in the application of the rule of &L1:

$$\frac{\underline{A}, C \vdash G}{\underline{A}, C\&D \vdash G}\ \&L1$$

then $\{\underline{A}, C\&D, \underline{U}\}$-$\{\underline{U}\} \vdash G$ is provable in IMALL*

$$\frac{\{\underline{A}, C, \underline{U}\}\text{-}\{\underline{U}\} \vdash G \qquad \overline{\underline{U} \subseteq \underline{A}, \underline{U}}}{\{\underline{A}, C\&D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}\ \&L1^*$$

By induction hypothesis if $\underline{A}$, $C \vdash G$ is provable in MALL then $\{\underline{A}, C, \underline{U}\}$-$\{\underline{U}\} \vdash G$ is provable in IMALL*.

Similarly for &L1.

Suppose that the proof ends in the application of the rule of $\oplus$R1:

$$\frac{\underline{A} \vdash C}{\underline{A} \vdash C\oplus D}\ \oplus R1$$

then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C\oplus D$ is provable in IMALL*

$$\frac{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C}{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C\oplus D}\ \oplus R1^{*}$$

By induction hypothesis if $\underline{A} \vdash C$ is provable in MALL then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C$ is provable in IMALL*.

Similarly for $\oplus$R2

Suppose that the proof ends in the application of the rule of 1L:

$$\frac{\underline{A} \vdash G}{\underline{A}, 1 \vdash \ \ G}\ 1L$$

then $\{\underline{A}, 1, \underline{U}\}\text{-}\{\underline{U}\} \vdash$ Gis provable in IMALL*

$$\frac{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}{\{\underline{A}, 1, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}\ 1L^{*}$$

By induction hypothesis if $\underline{A} \vdash G$ is provable in MALL then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL*.

Suppose that the proof ends in the application of the rule of $\otimes$L:

$$\frac{\underline{A}, C, D \vdash G}{\underline{A}, C\otimes D \vdash G}\ \otimes L$$

then $\{\underline{A}, C\otimes D, \underline{U}\}\text{-}\{\underline{U}\} \vdash$ Gis provable in IMALL*

$$\frac{\{\underline{A}, C, D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}{\{\underline{A}, C\otimes D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}\ \otimes L^{*}$$

By induction hypothesis if $\underline{A}, C, D \vdash G$ is provable in MALL then $\{\underline{A}, C, D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL*.

Suppose that the proof ends in the application of the rule of $\oplus$L:

$$\frac{\underline{A}, C \vdash G \quad \underline{A}, D \vdash G}{\underline{A}, C\oplus D \vdash G}\ \oplus L$$

then $\{\underline{A}, C\oplus D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL*

$$\frac{\{\underline{A}, C, \underline{U}\}\text{-}\{\underline{U}\} \vdash G \quad \{\underline{A}, D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G \quad \overline{\underline{U} \subseteq \underline{A}, \underline{U}}}{\{\underline{A}, C\oplus D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G}\ \oplus L^{*}$$

By induction hypothesis if $\underline{A}, C \vdash G$ and $\underline{A}, D \vdash G$ are provable in MALL then $\{\underline{A}, C, \underline{U}\}\text{-}\{\underline{U}\} \vdash G$ and $\{\underline{A}, D, \underline{U}\}\text{-}\{\underline{U}\} \vdash G$ are provable in IMALL*.

Suppose that the proof ends in the application of the rule of &R:

$$\frac{\underline{A} \vdash C \quad \underline{A} \vdash D}{\underline{A} \vdash C \& D}\ \&R$$

then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C \otimes D$ is provable in IMALL*

$$\frac{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C \quad \{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash D}{\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C \& D}\ \&R^{*}$$

By induction hypothesis if $\underline{A} \vdash C$ and $\underline{A} \vdash D$ are provable in MALL then $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash C$ and $\{\underline{A}, \underline{U}\}\text{-}\{\underline{U}\} \vdash D$ are provable in IMALL*.

Lemma 2.2: If $\vdash \{\underline{A}\}-\{\underline{U}\}$ is provable in IMALL* then $\vdash \underline{B}$, where $\underline{B}$ is the context which contains all formulae in $\underline{A}$, with one occurance of all the formulae in $\underline{U}$ removed, is provable in IMALL.

Proof: By induction on the height of the IMALL* proof:

Suppose that the last rule in the IMALL* proof is the axiom of I*

$$\frac{}{\{C, \underline{A}\}-\{\underline{A}\} \vdash C}\ I^*$$

then $C \vdash C$ is provable in IMALL

$$\frac{}{C \vdash C}\ I$$

Suppose that the last rule in the IMALL* proof is the axiom of 1*

$$\frac{}{\{\underline{A}\}-\{\underline{A}\} \vdash 1}\ 1^*$$

then $\vdash 1$ is provable in IMALL

$$\frac{}{\vdash 1}\ 1$$

Suppose that the last rule in the IMALL* proof is the axiom of 0*

$$\frac{}{\{\underline{A}, 0\}-\{\underline{U}\} \vdash G}\ 0^*\ (\underline{U} \text{ is a subset of } \underline{A})$$

then $\underline{B}, 0 \vdash G$ is provable in IMALL, where $\underline{B}$ is $\{\underline{A}\}-\{\underline{U}\}$

$$\frac{}{\underline{B}, 0 \vdash G}\ 0$$

Suppose that the last rule in the IMALL* proof is the axiom of T*

$$\frac{}{\{\underline{A}\}-\{\underline{U}\} \vdash T}\ T^*\ (\underline{U} \text{ is a subset of } \underline{A})$$

then $\underline{B} \vdash T$ is provable in IMALL, where $\underline{B}$ is $\{\underline{A}\}-\{\underline{U}\}$

$$\frac{}{\underline{B} \vdash T}\ T$$

Suppose that the last rule in the IMALL* proof is $\multimap R^*$

$$\frac{\{\underline{A}, C\}-\{\underline{U}\} \vdash D \quad \overline{\underline{U} \subseteq \underline{A}}}{\{\underline{A}\}-\{\underline{U}\} \vdash C \multimap D}\ \multimap R^*$$

then $\underline{B} \vdash C \multimap D$ is provable in IMALL, where $\underline{B}$ is $\{\underline{A}\}-\{\underline{U}\}$:

$$\frac{\underline{B}, C \vdash D}{\underline{B} \vdash C \multimap D} \multimap R$$

By induction hypothesis if $\{\underline{A}, C\}\text{-}\{\underline{U}\} \vdash D$ is provable in IMALL* then $\underline{B}, C \vdash D$ is provable in IMALL.

Suppose that the last rule in the IMALL* proof is &L1*

$$\frac{\{\underline{A}, C\}\text{-}\{\underline{U}\} \vdash G \quad \overline{\underline{U} \subseteq \underline{A}}}{\{\underline{A}, C\&D\}\text{-}\{\underline{U}\} \vdash G} \&L1^*$$

then $\underline{B}, C\&D \vdash G$ is provable in IMALL, where B is $\{\underline{A}\}\text{-}\{\underline{U}\}$.

$$\frac{\underline{B}, C \vdash G}{\underline{B}, C\&D \vdash G} \&L1$$

By induction hypothesis if $\{\underline{A}, C\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL* then $\underline{B}, C \vdash G$ is provable in IMALL.

Similarly for &L2*

Suppose that the last rule in the IMALL* proof is &R*

$$\frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \quad \{\underline{A}\}\text{-}\{\underline{U}\} \vdash D}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C\&D} \&R^*$$

then $\underline{B} \vdash C\&D$ is provable in IMALL, where B is $\{\underline{A}\}\text{-}\{\underline{U}\}$

$$\frac{\underline{B} \vdash C \quad \underline{B} \vdash D}{\underline{B} \vdash C\&D} \&R$$

By induction hypothesis if $\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C$ and $\{\underline{A}\}\text{-}\{\underline{U}\} \vdash D$ are provable in IMALL* then $\underline{B} \vdash C$ and $\underline{B} \vdash D$ are provable in IMALL.

Suppose that the last rule in the IMALL* proof is ⊕R1*

$$\frac{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C}{\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C \oplus D} \oplus R1^*$$

then $\underline{B} \vdash C \oplus D$ is provable in IMALL, where B is $\{\underline{A}\}\text{-}\{\underline{U}\}$

$$\frac{\underline{B} \vdash C}{\underline{B} \vdash C \oplus D} \oplus R1$$

By induction hypothesis if $\{\underline{A}\}\text{-}\{\underline{U}\} \vdash C$ is provable in IMALL* then $\underline{B} \vdash C$ is provable in IMALL.

Similarly for ⊕R2*

Suppose that the last rule in the IMALL* proof is 1L*

$$\frac{\{\underline{A}\}\text{-}\{\underline{U}\}\vdash G}{\{\underline{A}, 1\}\text{-}\{\underline{U}\}\vdash G}\ 1L^*$$

then $\underline{B} \vdash G$ is provable in IMALL, where B is $\{\underline{A}\}\text{-}\{\underline{U}\}$

$$\frac{\underline{B}\vdash G}{\underline{B}, 1\vdash G}\ 1L$$

By induction hypothesis if $\{\underline{A}\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL* then $\underline{B} \vdash G$ is provable in IMALL.

Suppose that the last rule in the IMALL* proof is ⊕L*

$$\frac{\{\underline{A}, C\}\text{-}\{\underline{U}\}\vdash G \qquad \{\underline{A}, D\}\text{-}\{\underline{U}\}\vdash G \qquad \overline{\underline{U}\subseteq\underline{A}}}{\{\underline{A}, C\oplus D\}\text{-}\{\underline{U}\}\vdash G}\ \oplus L^*$$

then $\underline{B}, C\oplus D \vdash G$ is provable in IMALL, where B is $\{\underline{A}\}\text{-}\{\underline{U}\}$

$$\frac{\underline{B}, C\vdash G \qquad \underline{B}, D\vdash G}{\underline{B}, C\oplus D\vdash G}\ \oplus L$$

By induction hypothesis if $\{\underline{A}, C\}\text{-}\{\underline{U}\} \vdash G$ and $\{\underline{A}, D\}\text{-}\{\underline{U}\} \vdash G$ are provable in IMALL* then $\underline{B}, C \vdash G$ and $\underline{B}, D \vdash G$ are provable in IMALL.

Suppose that the last rule in the IMALL* proof is ⊗L*

$$\frac{\{\underline{A}, C, D\}\text{-}\{\underline{U}\}\vdash G}{\{\underline{A}, C\otimes D\}\text{-}\{\underline{U}\}\vdash G}\ \otimes L^*$$

then $\underline{B}, C\otimes D \vdash G$ is provable in IMALL, where B is $\{\underline{A}\}\text{-}\{\underline{U}\}$

$$\frac{\underline{B}, C, D\vdash G}{\underline{B}, C\otimes D\vdash G}\ \otimes L$$

By induction hypothesis if $\{\underline{A}, C, D\}\text{-}\{\underline{U}\} \vdash G$ is provable in IMALL* then $\underline{B}, C, D \vdash G$ is provable in IMALL.

Suppose that the last rule in the IMALL* proof is ⊗R*

$$\frac{\{\underline{A}\}\text{-}\{\underline{U1}\}\vdash C \qquad \{\underline{U1}\}\text{-}\{\underline{U}\}\vdash D}{\{\underline{A}\}\text{-}\{\underline{U}\}\vdash C\otimes D}\ \otimes R^*$$

then $\underline{B}, C\otimes D \vdash G$ is provable in IMALL, where B is $\{\underline{A}\}\text{-}\{\underline{U}\}$

$$\frac{\underline{B1}\vdash C \qquad \underline{B2}\vdash D}{\underline{B2}, \underline{B2}\vdash C\otimes D}\ \otimes R$$

(Where $\underline{B1}$ and $\underline{B2}$ form a partition of $\underline{B}$)

$\{\underline{A}\}\text{-}\{\underline{U1}\}$ and $\{\underline{U1}\}$ form a partition of A
Therefore $\{\underline{A}\}\text{-}\{\underline{U1}\}$ and $\{\underline{U1}\}\text{-}\{\underline{U}\}$ form a partition of $\{\underline{A}\}\text{-}\{\underline{U}\}$.

By induction hypothesis if $\{\underline{A}\}-\{\underline{U1}\} \vdash C$ and $\{\underline{U1}\}-\{\underline{U}\} \vdash D$ are provable in IMALL* then $\underline{B1} \vdash C$ and $\underline{B2} \vdash D$ are provable in IMALL.

Suppose that the last rule in the IMALL* proof is $\multimap L^*$

$$\frac{\{\underline{A}\}-\{\underline{U1}\} \vdash C \qquad \{\underline{U1}, D\}-\{\underline{U}\} \vdash G \qquad \overline{\underline{U} \subseteq \underline{A}}}{\{C \multimap D, \underline{A}\}-\{\underline{U}\} \vdash G} \multimap L^*$$

then $C \multimap D, \underline{B} \vdash G$ is provable in IMALL, where B is $\{\underline{A}\}-\{\underline{U}\}$

$$\frac{\underline{B1} \vdash C \quad D, \underline{B2} \vdash G}{C \multimap D, \underline{B1}, \underline{B2} \vdash G} \multimap L \qquad \text{(Where } \underline{B1} \text{ and } \underline{B2} \text{ form a partition of } \underline{B}\text{)}$$

$\{\underline{A}\}-\{\underline{U1}\}$ and $\{\underline{U1}\}$ form a partition of A
Therefore $\{\underline{A}\}-\{\underline{U1}\}$ and $\{\underline{U1}\}-\{\underline{U}\}$ form a partition of $\{\underline{A}\}-\{\underline{U}\}$.

By induction hypothesis if $\{\underline{A}\}-\{\underline{U1}\} \vdash C$ and $\{\underline{U1}, D\}-\{\underline{U}\} \vdash G$ are provable in IMALL* then $\underline{B1} \vdash C$ and $D, \underline{B2} \vdash G$ are provable in IMALL.

4.3 MALL* lemmas

Lemma 3.1: If $\vdash \underline{A}, G$ is provable in MALL then $\underline{A}^\bot, G^\bot \vdash \bot$ and $\underline{A}^\bot \vdash G$ are provable in MALL* if negated atoms are considered to be atoms. A MALL formula of the form $C ⅋ D$ is considered to be equivalent to the IMALL formula $(C^\bot \multimap D)\&(D^\bot \multimap C)$.

Proof: By induction on the height of the MALL proof:

Suppose that the MALL proof ends in an application of the rule of I:

$$\frac{}{\vdash a, a^\bot}\,I$$

then

$a^\bot, a \vdash \bot$,
$a \vdash a$ and
$a^\bot \vdash a^\bot$
are provable in MALL* by the rules of I*, I and I respectively.

Suppose that the MALL proof ends in an application of the rule of 1:

$$\frac{}{\vdash 1}\,1$$

then

$\bot \vdash \bot$ and
$\vdash 1$

are provable in MALL* by the rules of I and 1 respectively.

Suppose that the MALL proof ends in an application of the rule of $\top$:

$$\frac{}{\vdash \top, \underline{A}, G}\,\top$$

then

$0, \underline{A}^\bot, G^\bot \vdash \bot$ and
$0, \underline{A}^\bot \vdash G$
$G^\bot, \underline{A}^\bot \vdash \top$
are provable in MALL* by the rule of 0, 0 and $\top$ respectively.

Supposing that the MALL proof ends in an application of the rule of $\bot$:

$$\frac{\vdash \underline{A}, G}{\vdash \bot, \underline{A}, G}\,\bot$$

then

$1, \underline{A}^\bot, G \vdash \bot$
$1, \underline{A}^\bot \vdash G$ and
$G^\bot, \underline{A}^\bot \vdash \bot$
are provable in MALL*:

$$\dfrac{\underline{A}^{\perp},\ G^{\perp} \vdash \bot}{\underline{A}^{\perp}, 1, G^{\perp} \vdash \bot}\ 1L \qquad\qquad \dfrac{\underline{A}^{\perp} \vdash G}{\underline{A}^{\perp}, 1 \vdash G}\ 1L$$

By induction hypothesis if $\vdash \underline{A}, G$ is provable in MALL then
$G^{\perp}, \underline{A}^{\perp} \vdash \bot$ and
$\underline{A}^{\perp} \vdash G$
are provable in MALL*.

Supposing that the MALL proof ends in an application of the rule of $\otimes$:

Case 1:

$$\dfrac{\vdash C, \underline{A1} \quad \vdash D, \underline{A2}, G}{\vdash C \otimes D, \underline{A1}, \underline{A2}, G}\ \otimes$$

then

$((C^{\perp} \multimap D) \oplus (D^{\perp} \multimap C))^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp}, G^{\perp} \vdash \bot$,
$((C^{\perp} \multimap D) \oplus (D^{\perp} \multimap C))^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp} \vdash G$ and
$\underline{A1}^{\perp}, \underline{A2}^{\perp}, G^{\perp} \vdash C \otimes D$
are provable in MALL*:

$$\dfrac{\dfrac{\underline{A1}^{\perp} \vdash C \quad D^{\perp}, \underline{A2}^{\perp},\ G^{\perp} \vdash \bot}{C \multimap D^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp},\ G^{\perp} \vdash \bot}\ \multimap L}{(C \multimap D^{\perp}) \& (D \multimap C^{\perp}), \underline{A1}^{\perp}, \underline{A2}^{\perp},\ G^{\perp} \vdash \bot}\ \&L1$$

$$\dfrac{\dfrac{\underline{A1}^{\perp} \vdash C \quad D^{\perp}, \underline{A2}^{\perp} \vdash G}{C \multimap D^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp} \vdash G}\ \multimap L}{(C \multimap D^{\perp}) \& (D \multimap C^{\perp}), \underline{A1}^{\perp}, \underline{A2}^{\perp} \vdash G}\ \&L1$$

$$\dfrac{\underline{A1}^{\perp} \vdash C \quad \underline{A2}^{\perp}, G^{\perp} \vdash D}{\underline{A1}^{\perp}, \underline{A2}^{\perp}, G^{\perp} \vdash C \otimes D}\ \otimes R$$

By induction hypothesis if $\vdash C, \underline{A1}$ is provable in MALL then $\underline{A1}^{\perp} \vdash C$ is provable in MALL*.

By induction hypothesis if $\vdash D, \underline{A2}$ is provable in MALL then
$D^{\perp}, \underline{A2}^{\perp} \vdash G$,
$D^{\perp}, \underline{A2}^{\perp}, G^{\perp} \vdash \bot$ and
$\underline{A2}^{\perp}, G^{\perp} \vdash D$
are provable in MALL*.

Case 2:

$$\dfrac{\vdash C, \underline{A1}, G \quad \vdash D, \underline{A2}}{\vdash C \otimes D, \underline{A1}, \underline{A2}, G}\ \otimes$$

then

$((C^{\perp} \multimap D) \oplus (D^{\perp} \multimap C))^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp}, G^{\perp} \vdash \bot$
$((C^{\perp} \multimap D) \oplus (D^{\perp} \multimap C))^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp} \vdash G$,
$\underline{A1}^{\perp}, \underline{A2}^{\perp}, G^{\perp} \vdash C \otimes D$ and
are provable in MALL*:

$$\dfrac{\dfrac{\underline{A1}^{\perp},\ G^{\perp} \vdash C \qquad D^{\perp}, \underline{A2}^{\perp} \vdash \perp}{C \multimap D^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp},\ G^{\perp} \vdash \perp}\ \multimap L}{(C \multimap D^{\perp}) \& (D \multimap C^{\perp}), \underline{A1}^{\perp}, \underline{A2}^{\perp},\ G^{\perp} \vdash \perp}\ \& L1$$

$$\dfrac{\dfrac{\underline{A2}^{\perp} \vdash D \qquad C^{\perp}, \underline{A1}^{\perp} \vdash G}{D \multimap C^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp} \vdash G}\ \multimap L}{(C \multimap D^{\perp}) \& (D \multimap C^{\perp}), \underline{A1}^{\perp}, \underline{A2}^{\perp} \vdash G}\ \& L2$$

$$\dfrac{\underline{A1}^{\perp}, G^{\perp} \vdash C \qquad \underline{A2}^{\perp} \vdash D}{\underline{A1}^{\perp}, \underline{A2}^{\perp}, G^{\perp} \vdash C \otimes D}\ \otimes R$$

By induction hypothesis if $\vdash C, \underline{A1}, G$ is provable in MALL then
$\underline{A1}^{\perp}, G^{\perp} \vdash C$,
$C^{\perp}, \underline{A1}^{\perp} \vdash G$ and
$\underline{A1}^{\perp}, G^{\perp} \vdash C$
are provable in MALL*.

By induction hypothesis if $\vdash D, \underline{A2}$ is provable in MALL then
$\underline{A2}^{\perp} \vdash D$ and
$\underline{A1}^{\perp}, G^{\perp} \vdash C$
are provable in MALL*.

Supposing that the MALL proof ends in an application of the rule of $1\oplus$:

$$\dfrac{\vdash C, \underline{A}, G}{\vdash C \oplus D, \underline{A}, G}\ 1\oplus$$

then

$C^{\perp} \& D^{\perp}, \underline{A}^{\perp}, G^{\perp} \vdash \perp$,
$C^{\perp} \& D^{\perp}, \underline{A}^{\perp} \vdash G$ and
$\underline{A}^{\perp}, G^{\perp} \vdash C \oplus D$
are provable in MALL*:

$$\dfrac{\underline{A}^{\perp}, C^{\perp},\ G \vdash \perp}{\underline{A}^{\perp}, C^{\perp} \& D^{\perp},\ G \vdash \perp}\ \& L1 \qquad \dfrac{\underline{A}^{\perp}, C^{\perp} \vdash G}{\underline{A}^{\perp}, C^{\perp} \& D^{\perp} \vdash G}\ \& L1$$

$$\dfrac{\underline{A}^{\perp}, G^{\perp} \vdash C}{\underline{A}^{\perp}, G^{\perp} \vdash C \oplus D}$$

By induction hypothesis if $\vdash C, \underline{A}, G$ is provable in MALL then
$\underline{A}^{\perp}, C^{\perp}, G \vdash \perp$,
$\underline{A}^{\perp}, C^{\perp} \vdash G$ and
$\underline{A}^{\perp}, C^{\perp}, G \vdash \perp$
are provable in MALL*.

Similarly for $2\oplus$.

Suppose that the MALL proof ends in an application of the rule of &:

$$\dfrac{\vdash C, \underline{A}, G \qquad \vdash D, \underline{A}, G}{\vdash C \& D, \underline{A}, G}\ \&$$

Then

$C^\perp \oplus D^\perp, \underline{A}^\perp, G^\perp \vdash \perp$,
$C^\perp \oplus D^\perp, \underline{A}^\perp \vdash G$ and
$\underline{A}^\perp, G^\perp \vdash C \& D$
are provable in MALL*

$$\frac{\underline{A}^\perp, C^\perp, G^\perp \vdash \perp \quad \underline{A}^\perp, D^\perp, G^\perp \vdash \perp}{\underline{A}^\perp, C^\perp \oplus D^\perp, G^\perp \vdash \perp}\ \oplus L \qquad \frac{\underline{A}^\perp, C^\perp \vdash G \quad \underline{A}^\perp, D^\perp \vdash G}{\underline{A}^\perp, C^\perp \oplus D^\perp \vdash G}\ \oplus L$$

$$\frac{\underline{A}^\perp, G^\perp \vdash C \quad \underline{A}^\perp, G^\perp \vdash D}{\underline{A}^\perp, G^\perp \vdash C \& D}\ \& R$$

By induction hypothesis if $\vdash C, \underline{A}, G$ is provable in MALL then
$\underline{A}^\perp, C^\perp, G^\perp \vdash \perp$,
$\underline{A}^\perp, C^\perp \vdash G$ and
$\underline{A}^\perp, G^\perp \vdash C$
are provable in MALL*.

By induction hypothesis if $\vdash D, \underline{A}, G$ is provable in MALL then
$\underline{A}^\perp, D^\perp, G^\perp \vdash \perp$,
$\underline{A}^\perp, D^\perp \vdash G$ and
$\underline{A}^\perp, G^\perp \vdash D$
are provable in MALL*.

Suppose that the MALL proof ends in an application of the rule of $\parr$:

$$\frac{\vdash C, D, \underline{A}, G}{\vdash C \parr D, \underline{A}, G}\ \parr$$

Then

$\underline{A}^\perp, C^\perp \otimes D^\perp, G^\perp \vdash \perp$,
$\underline{A}^\perp, C^\perp \otimes D^\perp \vdash G$ and
$\underline{A}^\perp, G^\perp \vdash (C^\perp \multimap D) \& (C \multimap D^\perp))$
are provable in MALL*

$$\frac{\underline{A}^\perp, C^\perp, D^\perp \vdash G}{\underline{A}^\perp, C^\perp \otimes D^\perp \vdash G}\ \otimes L$$

$$\frac{\underline{A}^\perp, C^\perp, D^\perp, G^\perp \vdash \perp}{\underline{A}^\perp, C^\perp \otimes D^\perp, G^\perp \vdash \perp}\ \otimes L$$

$$\frac{\dfrac{\underline{A}^\perp, G^\perp, \ C^\perp \vdash D}{\underline{A}^\perp, G^\perp \vdash (C^\perp \multimap D)}\ \multimap R}{\underline{A}^\perp, G^\perp \vdash (C^\perp \multimap D) \& (C \multimap D^\perp)}\ \otimes L$$

By induction hypothesis if $\vdash C, D, \underline{A}, G$ is provable in MALL then
$\underline{A}^\perp, C^\perp, D^\perp \vdash G$,
$\underline{A}^\perp, C^\perp, D^\perp, G^\perp \vdash \perp$ and
$\underline{A}^\perp, G^\perp, C^\perp \vdash D$
are provable in MALL*.

Lemma 3.2: If $\underline{A} \vdash G$ is provable in MALL* then $\vdash \underline{A}^{\perp}, G$ is provable in MALL.

Proof: By induction on the height of the MALL* proof.

Suppose that the last rule applied in the MALL* proof is I*:

$$\dfrac{}{a, a^{\perp} \vdash \perp}\ \mathrm{I}^{*}$$

then $\vdash a^{\perp}, a, \perp$ is provable in MALL:

$$\dfrac{\dfrac{}{\vdash a^{\perp}, a}\ \mathrm{I}}{\vdash a^{\perp}, a, \perp}\ \perp$$

Suppose that the last rule applied in the MALL* proof is I:

$$\dfrac{}{a \vdash a}\ \mathrm{I}$$

then $\vdash a^{\perp}, a$ is provable in MALL by the I axiom.

Suppose that the last rule applied in the MALL* proof is 1:

$$\dfrac{}{\vdash 1}\ 1$$

then $\vdash 1$ is provable in MALL by the 1 axiom

Suppose that the last rule applied in the MALL* proof is 0:

$$\dfrac{}{\underline{A}, 0 \vdash G}\ 0$$

then $\vdash \underline{A}^{\perp}, \top, G$ is provable in MALL by the $\top$ axiom

Suppose that the last rule applied in the MALL* proof is $\top$:

$$\dfrac{}{\underline{A} \vdash \top}\ \top$$

then $\vdash \underline{A}^{\perp}, \top$ is provable in MALL by the $\top$ axiom

Suppose that the last rule applied in the MALL* proof is 1L:

$$\dfrac{\underline{A} \vdash G}{\underline{A}, 1 \vdash G}\ 1\mathrm{L}$$

then $\vdash \underline{A}^{\perp}, \perp, G$ is provable in MALL:

$$\dfrac{\vdash \underline{A}^{\perp}, G}{\vdash \underline{A}^{\perp}, \perp, G}\ \top$$

By induction hypothesis if $\underline{A} \vdash G$ is provable in MALL* then $\vdash \underline{A}^{\perp}, G$ is provable in MALL

Suppose that the last rule applied in the MALL* proof is $\multimap$L:

$$\frac{\underline{A1} \vdash C \quad D, \underline{A2} \vdash G}{C \multimap D, \underline{A1}, \underline{A2} \vdash G} \multimap L$$

then $\vdash C \otimes D^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp}, G$ is provable in MALL:

$$\frac{\vdash C, \underline{A1}^{\perp} \quad \vdash D^{\perp}, \underline{A2}^{\perp}, G}{\vdash C \otimes D^{\perp}, \underline{A1}^{\perp}, \underline{A2}^{\perp}} \otimes$$

By induction hypothesis if $\underline{A1} \vdash C$ and $D, \underline{A2} \vdash G$ are provable in MALL* then
$\vdash C, \underline{A1}^{\perp}$ and
$\vdash D^{\perp}, \underline{A2}^{\perp}, G$
are provable in MALL

Suppose that the last rule applied in the MALL* proof is &R:

$$\frac{\underline{A} \vdash C \quad \underline{A} \vdash D}{\underline{A} \vdash C \& D} \& R$$

then $\vdash \underline{A}^{\perp}, C \& D$ is provable in MALL:

$$\frac{\vdash C, \underline{A}^{\perp} \quad \vdash D, \underline{A}^{\perp}}{\vdash C \& D, \underline{A}^{\perp}} \&$$

By induction hypothesis if $\underline{A} \vdash C$ and $\underline{A} \vdash D$ are provable in MALL* then
$\vdash C, \underline{A}^{\perp}$ and
$\vdash D, \underline{A}^{\perp}$
are provable in MALL

Suppose that the last rule applied is $\multimap$R

$$\frac{\underline{A}, C \vdash D}{\underline{A} \vdash C \multimap D} \multimap R$$

then $\vdash \underline{A}^{\perp}, C^{\perp} \parr D$ is provable in MALL

$$\frac{\vdash C^{\perp}, D, \underline{A}^{\perp}}{\vdash C^{\perp} \parr D, \underline{A}^{\perp}} \parr$$

By induction hypothesis if $\underline{A}, C \vdash D$ is provable in MALL* then $\vdash C^{\perp}, D, \underline{A}^{\perp}$ is provable in MALL

Suppose that the last rule applied is $\oplus$L

$$\frac{\underline{A}, C \vdash G \quad \underline{A}, D \vdash G}{\underline{A}, C \oplus D \vdash G} \oplus L$$

then $\vdash \underline{A}^{\perp}, C^{\perp} \& D^{\perp}, G$ is provable in MALL

$$\frac{\vdash C^{\perp}, \underline{A}^{\perp}, G \quad \vdash D^{\perp}, \underline{A}^{\perp}, G}{\vdash C^{\perp} \& D^{\perp}, \underline{A}^{\perp}, G} \&$$

By induction hypothesis if $\underline{A}$, $C \vdash G$ are $\underline{A}$, $D \vdash G$ provable in MALL* then
$\vdash C^{\perp}, \underline{A}^{\perp}, G$ and
$\vdash D^{\perp}, \underline{A}^{\perp}, G$
are provable in MALL

Suppose that the last rule applied is $\oplus$R1:

$$\frac{\underline{A} \vdash C}{\underline{A} \vdash C \oplus D}\ \oplus R1$$

then $\vdash \underline{A}^{\perp}, C \oplus D$ is provable in MALL

$$\frac{\vdash C, \underline{A}^{\perp}}{\vdash C \oplus D, \underline{A}^{\perp}}\ 1\oplus$$

By induction hypothesis if $\underline{A} \vdash C$ is provable in MALL* then $\vdash C, \underline{A}^{\perp}$ is provable in MALL

Similarly for $\oplus$R2

Suppose that the last rule applied is &L1

$$\frac{\underline{A}, C \vdash G}{\underline{A}, C \& D \vdash G}\ \&L1$$

then $\vdash \underline{A}^{\perp}, C^{\perp} \oplus D^{\perp}, G$ is provable in MALL

$$\frac{\vdash C^{\perp}, \underline{A}^{\perp}, G}{\vdash C^{\perp} \oplus D^{\perp}, \underline{A}^{\perp}, G}\ 1\oplus$$

By induction hypothesis if $\underline{A}$, $C \vdash G$ is provable in MALL* then $\vdash C^{\perp}, \underline{A}^{\perp}, G$ is provable in MALL

Similarly for &L2

Suppose that the last rule applied is $\otimes$L

$$\frac{\underline{A}, C, D \vdash G}{\underline{A}, C \otimes D \vdash G}\ \otimes L$$

then $\vdash \underline{A}^{\perp}, C^{\perp} \parr D^{\perp}, G$ is provable in MALL

$$\frac{\vdash C^{\perp}, D^{\perp}, \underline{A}^{\perp}, G}{\vdash C^{\perp} \parr D^{\perp}, \underline{A}^{\perp}, G}\ \parr$$

By induction hypothesis if $\underline{A}$, $C, D \vdash G$ is provable in MALL* then $\vdash C^{\perp}, D^{\perp}, \underline{A}^{\perp}, G$ is provable in MALL

Suppose that the last rule applied is ⊗R

$$\frac{\underline{A1} \vdash C \quad \underline{A2} \vdash D}{\underline{A2}, \underline{A2} \vdash C \otimes D} \otimes R$$

then $\vdash \underline{A1}^{\perp}, \underline{A2}^{\perp}, C \otimes D$ is provable in MALL

$$\frac{\vdash C, \underline{A1}^{\perp} \quad \vdash D, \underline{A2}^{\perp}}{\vdash C \otimes D, \underline{A1}^{\perp}, \underline{A2}^{\perp}} \otimes$$

By induction hypothesis if $\underline{A1} \vdash C$ and $\underline{A2} \vdash D$ provable in MALL* then
$\vdash C, \underline{A1}^{\perp}$ and
$\vdash D, \underline{A2}^{\perp}$
are provable in MALL

4.4 ILL* Lemma

Lemma 4.1: If $\{A\}\text{-}\{U\} \vdash G$ is provable in ILL* then $\underline{B} \vdash G$, where $\underline{B}$ is the context that contains all the formula in $\underline{A}$ with one occurrence of each formula in $\underline{U}$ removed, is provable in ILL.

Proof: By induction on the height of the ILL* proof.

For the cases where the last rule applied in the ILL* proof is a non modal operator, refer to lemma 2.1.

Suppose that the ILL* proof ends in an application of the rule of !R*:

$$\frac{\{!\underline{A1}\}\text{-}\ \underline{\{U\}} \vdash G}{\{!\underline{A1},\ \underline{A2}\}\text{-}\{\underline{U},\ \underline{A2}\} \vdash\ !G}\ !R$$

then then $\underline{B} \vdash\ !G$ is provable in ILL, where $\underline{B}$ is $\{!\underline{A1}\}\text{-}\{\underline{U}\}$ (By definition $\{!\underline{A1},\ \underline{A2}\}\text{-}\{\underline{U},\ \underline{A2}\} = \{!\underline{A1}\}\text{-}\{\underline{U}\}$):

$$\frac{\underline{B} \vdash G}{\underline{B} \vdash\ !G}\ !R$$

By induction hypothesis if $\{!\underline{A1}\}\text{-}\ \underline{\{U\}} \vdash G$ is provable in MALL* then $\underline{B} \vdash G$ is provable in MALL.

Suppose that the ILL* proof ends in an application of the rule of !D*:

$$\frac{\{\underline{A},\ C\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{\underline{A},\ !C\}\text{-}\{\underline{U}\} \vdash G}\ !D^*$$

then $!C, \underline{B} \vdash G$ is provable in MALL where $\underline{B}$ is $\{\underline{A}\}\text{-}\{\underline{U}\}$:

$$\frac{\underline{B},\ C \vdash G}{\underline{B},\ !C \vdash G}\ !D$$

By induction hypothesis if $\{\underline{A},\ C\}\text{-}\{\underline{U}\} \vdash G$ is provable in MALL* then $\underline{B},\ C \vdash G$ is provable in MALL.

Suppose that the ILL* proof ends in an application of the rule of !C*:

$$\frac{\{!C(0),\ !C(N\text{-}1),\ \underline{A}\}\text{-}\{\underline{U}\} \vdash G \quad \underline{U} \subseteq \underline{A}}{\{!C(N),\ \underline{A}\}\text{-}\{\underline{U}\} \vdash G}\ !C \qquad (N \geq 0)$$

then $!C,\ !C,\ \underline{B} \vdash G$ is provable in MALL where $\underline{B}$ is $\{\underline{A}\}\text{-}\{\underline{U}\}$:

$$\frac{\underline{A},\ !C,\ \ !C \vdash G}{\underline{A},\ \ !C \vdash G}\ C!$$

By induction hypothesis if $\{!C,\ !C,\ \underline{A}\}\text{-}\{\underline{U}\} \vdash G$ is provable in MALL* then $\underline{A},\ !C,\ \ !C \vdash G$ is provable in MALL.

REFERENCES

[1] S Abramsky, Computational Interpretations of Linear Logic, Draft.

[2] Apple™ Human Interface guidelines: The Apple Desktop Interface, Addison-Weseley 1987.

[3] R. L Constable, S. F Allen, H. M Bromley, W. R Cleaveland, J. F Cremer, R. W Harper, D. J Howe, T. B Knoblock, N. P Mendler, P. Panangaden, J. T Sasaki and S. F Smith, Implementing Mathematics with Nuprl Proof Development System, *Prentice Hall,* 1986.

[4] Roy Dyckhoff, Implementing a simple proof assistant, Proceedings of the workshop on Programing for Logic Teaching, no 23.88, 1987.

[5] Roy Dyckhoff and Neil Leslie, Logic Representation in MacProlog, *Proceedings of the Workshop on Programming Logic,* No54:100-109, 1989.

[6] J-Y. Girard, Linear Logic, *Theoretical Computer Science,* 50:1-102, 1987.

[7] J-Y. Girard, Y. Lafont, Linear Logic and Lazy Computation, *Lecture Notes in Computer Science no250,* Springer-Verlag, 1987.

[8] J-Y. Girard, Y. Lafont, and P.Taylor, Proofs and Types, *Cambridge Tracts in Theoretical Computer Science,* Cambridge University Press, 1989.

[9] Hodas and Miller, Logic Programming in a Fragment of Linear Logic, Draft.

[10] Lincoln, Mitchell, Scedrov and Shankar, Decision Problems for Propositional Linear Logic, *Proceedings of the 31st I.E.E.E Symposium on Foundations of Computer Science.*

[11] Lincoln, Mitchell, Scedrov and Shankar, Linearizing Intuitionistic Implication, Draft.

[12] Nariso Mari-Oliet and Jose Meseguer, From Petri Nets to Linear Logic, *Lecture Notes in Computer Science no389,* Sprin, 1989.

APPENDIX: THE SOURCE CODE

TRANSFORMATIONS

```
transformation('MALL', transform).
transformation('CLL', transform).
transformation('ILinCLL', transform).
transformation('ILinILL', translate).
transformation("IILinIMALL' ', astransform_problems).

initial_transformation('CLL', change_to_onesided_sequent).
initial_transformation('MALL', change_to_onesided_sequent).
initial_transformation('ILinCLL', translate_il_cll).
initial_transformation('ILinILL', translate_il_ill).
initial_transformation("IILinIMALL' ', translate_il_imall).

%++++++++++++++++++++++++++++++Initial Transformations

initial_transform(Logic, Problem, NewProblem) :-
        initial_transformation(Logic, Op), !,
        Op(Problem, NewProblem), !.
initial_transform(_, Problem, Problem).

change_to_onesided_sequent(Assumptions ⊢ Goals, ⊢ Problem) :-
        map(negate, Assumptions, X),
        append(X, Goals, Problem).

translate_il_cll(Assumtions ⊢ Goal, ⊢ Problem) :-
        mark_il(Goal, NewGoal),
        map(mark_il, Assumtions, Assumtions1),
        map(assumption_query, Assumtions1, NewAssumtions),
        append(NewAssumtions, [NewGoal], Problem).

translate_il_cll(Assums ⊢ G , ⊢ Newproblem) :-
  map(mark_il, G, NewGoal) ,
  map(mark_il, Assums, NAS) ,
  map(assumption_query , NAS , NewAssums) ,
  append(NewAssums , NewGoal , Newproblem1 ).

translate_il_ill(Assumtions ⊢ Goal, NewAssumtions ⊢ NewGoal) :-
       mark_il(Goal, NewGoal),
       map(mark_il, Assumtions, Assumtions1),
       map(assumption_bang, Assumtions1, NewAssumtions).

assumption_query(Formula, ? Formula ⊥).
assumption_bang(Formula, ! Formula).

%++++++++++++++++++++++++++++++++On the fly transformations

transform_problems(Logic, Problems, Newproblems) :-
        transformation(Logic, Op), !,
        Op(Problems, Newproblems).
transform_problems(_, Problems, Problems) :- !.
```

```
transform(Problems, Newproblems) :-
        map(transform_problem, Problems, Newproblems), !.


transform(⊢ A, ⊢ B) :- map(transform_cll_formula, A, B).
transform_cll_formula((⟦ A ⟧) ⊥, NewA1) :- !,
      translate_and_negate_il_cll(A, NewA1).
transform_cll_formula(⟦A⟧, NewA) :- !,
      translate_il_ll(A, NewA).
transform_cll_formula(Formula⊥, NewFormula) :- !,
      negate(Formula, NewFormula).
transform_cll_formula(A⊥ ⊸ B, (A ⅋ B)) :- !.
transform_cll_formula(A ⊸ B, ((A⊥) ⅋ B)) :- !.
transform_cll_formula(A, A).
```

The predicate negate is used to take advantage of Prologs' first argument indexing.

```
negate('1', '⊥') :- !.
negate('⊥', '1') :- !.
negate('T', '0') :- !.
negate('0', 'T') :- !.

negate((A⊥ ⊗ B⊥), A ⅋ B) :- !.
negate((A⊥ ⊗ B), A ⅋ B⊥) :- !.
negate((A ⊗ B⊥), A⊥ ⅋ B) :- !.
negate((A ⊗ B), A⊥ ⅋ B⊥) :- !.

negate((A⊥ & B⊥), A ⊕ B) :- !.
negate((A⊥ & B), A ⊕ B⊥) :- !.
negate((A & B⊥), A⊥ ⊕ B) :- !.
negate((A & B), A⊥ ⊕ B⊥) :- !.
negate((!A⊥), ?(A)) :- !.
negate((!A), ?(A⊥)) :- !.

negate((?A⊥), !(A)) :- !.
negate((?A), !(A⊥)) :- !.

negate((A⊥ ⅋ B⊥), A ⊗ B) :- !.
negate((A⊥ ⅋ B), A ⊗ B⊥) :- !.
negate((A ⅋ B⊥), A⊥ ⊗ B) :- !.
negate((A ⅋ B), A⊥ ⊗ B⊥) :- !.

negate((A⊥ ⊕ B⊥), A & B) :- !.
negate((A⊥ ⊕ B), A & B⊥) :- !.
negate((A ⊕ B⊥), A⊥ & B) :- !.
negate((A ⊕ B), A⊥ & B⊥) :- !.

negate((A ⊸ B⊥), A ⊗ B) :- !.
negate((A ⊸ B), A ⊗ B⊥) :- !.

negate(((A)⊥), A) :- !.
negate(A, A⊥) :- atom(A), !.
```

```
translate_il_ll(A & B, TA & TB) :-
      mark_il(A, TA),
      mark_il(B, TB).
translate_il_ll(A∨B, !TA ⊕ !TB) :-
      mark_il(A, TA),
      mark_il(B, TB).
translate_il_ll(A → B, !TA ⊗ TB) :-
      mark_il(A, TA),
      mark_il(B, TB).
translate_il_ll(¬A, !TA ⊗ 'T') :-
      mark_il(A, TA).
translate_il_ll(⋀, '0').
translate_il_ll(A, A ) :- atom(A).

translate_and_negate_il_cll(F, NF) :-
      translate_il_ll(F, F1),
      negate(F1, NF).

mark_il(A, A) :- atomic(A).
mark_il(A, ⟦A⟧).

% ILinILL translation
translate(Problems, Newproblems) :-
      map(translate_problem, Problems, Newproblems).

translate_problem(A ⊢ G, NA ⊢ NG) :-
      map(translate_il_ill,A, NA),
      translate_il_ill(G, NG).

translate_il_ill(⟦A⟧, NA) :- !,
      translate_il_ll(A, NA).
translate_il_ill(A,A).

% IILinIMALL

astransform_problems(Problems, NewProblems) :-
  map(astransform_sequent, Problems, NewProblems).

astransform_sequent(A ⊢ B, NA ⊢ NB) :-
  map(astransform, A, NA),
  astransform(B, NB ).

astransform(⟦A→B⟧+, k⊗(⟦A⟧- ⊸(k⊸⟦B⟧+)))  :- !.
astransform(⟦A⟧+, k⊗(NA⊕'⊥'⊗'T')) :- atom_transform(A , NA),! .

astransform(⟦(A→B)→C⟧-,k⊸(((⟦B→C⟧- ⊸(k⊸⟦A→B⟧+))⊸(k⊗'⊥'))⊕(k⊗⟦C⟧-)))  :- !.
astransform(⟦P→A⟧-, k⊸((k⊸(⟦P⟧+)⊸(k⊗'⊥'))⊕(k⊗⟦A⟧-)))  :-
  atom(P), !.
astransform(⟦A⟧-, NA) :- atom_transform(A, NA), !.

astransform(P, P).

atom_transform(⋀ , '⊥') :- !.
atom_transform(A , A) :- atom(A) .
```

```
% all

translate_il_imall(Problem, NewProblem) :-
  implicize_sequent(Problem, Problem1) ,
  depth_reduce(Problem1, Problem2),
  mark_ill(Problem2, NewProblem).

mark_ill(A ⊢ G, [k | NA] ⊢ ⟦G⟧+) :-
      map(mark_ill_formula, A, NA).

mark_ill_formula(A, ⟦(A)⟧-).

implicize_sequent(A ⊢ G, NA ⊢ NG) :-
  map(implicize_formula, A, NA),
  implicize(G, NG).

implicize_formula(A → B, NA → NB) :- !,
  implicize_formula(A, NA),
  implicize_formula(B, NB) .
implicize_formula(A ∨ B, (NB → NA) → NA) :- !,
  implicize_formula(A, NA),
  implicize_formula(B, NB) .
implicize_formula(¬ A,  NA → ⋀) :- !,
  implicize_formula(A, NA) .
implicize_formula(A, A) .

depth_reduce(As ⊢ G, NA ⊢ NG) :-
   remove((A→B)→(C→D), As, NA1),
   depth_reduce(['X'→(C→D), (A→B)→'X' | NA1] ⊢ G, NA ⊢ NG) .
depth_reduce(As ⊢ G, NA ⊢ NG) :-
   remove(P→((A→B)→C), As, NA1), atom(P),
   depth_reduce([P→('X'→C), (A→B)→'X' | NA1] ⊢ G, NA ⊢ NG) .
depth_reduce(As ⊢ G, NA ⊢ NG) :-
   remove(P→(A→(B →C)), As, NA1), atom(P),
   depth_reduce(['X'→(B→C), P→(A→'X') | NA1] ⊢ G, NA ⊢ NG) .
depth_reduce(As ⊢ G, NA ⊢ NG) :-
   remove(((A→B)→C)→P, As, NA1), atom(P),
   depth_reduce(['X'→(A→B), ('X'→C)→P | NA1] ⊢ G, NA ⊢ NG) .
depth_reduce(As ⊢ G, NA ⊢ NG) :-
   remove((A→(B→C))→P, As, NA1), atom(P),
   depth_reduce([(B→C)→'X', (A→'X')→P | NA1] ⊢ G, NA ⊢ NG) .
depth_reduce(As ⊢ (A→B)→(C→D), NA ⊢ NG) :-
   depth_reduce([(C→D)→'X' | As] ⊢ (A→B)→'X', NA ⊢ NG) .
depth_reduce(As ⊢  P→(A→(B →C)), NA ⊢ NG) :-
  atom(P),
   depth_reduce([(B→C)→'X' | As] ⊢ P→(A→'X'), NA ⊢ NG) .
depth_reduce(As ⊢  P→((A→B)→C), NA ⊢ NG) :-
  atom(P),
   depth_reduce(['X'→(B→C)  | As] ⊢ P→('X'→C), NA ⊢ NG) .
depth_reduce(As ⊢ (A→(B→C))→P, NA ⊢ NG) :-
  atom(P),
   depth_reduce(['X'→(B→C)  | As] ⊢ (A→'X')→P, NA ⊢ NG) .
depth_reduce(As ⊢ ((A→B)→C)→P, NA ⊢ NG) :-
  atom(P),
   depth_reduce([(B→C)→'X'  | As] ⊢ ('X'→C)→P, NA ⊢ NG) .
depth_reduce(A, A) .
```

TACTICS AND AXIOMS

```
initial_sequent_type('CLL', list ⊢ list).
initial_sequent_type('ILL', list ⊢ formula).
initial_sequent_type('MALL', list ⊢ list).
initial_sequent_type('IMALL', list ⊢ formula).
initial_sequent_type('ILinCLL', list ⊢ formula).
initial_sequent_type('ILinILL', list ⊢ formula).
initial_sequent_type('IL', list ⊢ formula).

% MALL

'MALL' tactic_of 'Cut'
converts ⊢ Problem into
⊢ A :: A1 and
⊢ A ⊥ :: A2
given Problem equ A1 plus A2 and A is_cut_formula.

'MALL' tactic_of '⊥'
converts ⊢ Prob where ⊥ ∈ Prob into
⊢ Prob - ⊥ .

'MALL' tactic_of ⊗
converts ⊢ Prob where C ⊗ D ∈ Prob into
⊢ C :: A1 and
⊢ D :: A2
given (Prob - C ⊗ D) equ A1 plus A2.

'MALL' tactic_of &
converts ⊢ Prob where C & D ∈ Prob into
⊢ [C / C & D]^Prob and
⊢ [D / C & D]^Prob.

'MALL' tactic_of ⅋
converts ⊢ Prob where C ⅋ D ∈ Prob into
⊢ [(C, D) / C ⅋ D]^Prob.

'MALL' tactic_of '2⊕'
converts ⊢ Prob where C ⊕ D ∈ Prob into
⊢ [D / C ⊕ D]^Prob.

'MALL' tactic_of '1⊕'
converts ⊢ Prob where C ⊕ D ∈ Prob into
⊢ [C / C ⊕ D]^Prob.

% CLL

'CLL' is_an_extension_of 'MALL'.

'CLL' tactic_of 'D?'
converts ⊢ Prob where ? C ∈ Prob into
⊢ [C / ? C]^Prob.

'CLL' tactic_of 'W?'
converts ⊢ Prob where ? C ∈ Prob into
⊢ Prob - ? C.
```

```
'CLL' tactic_of 'C?'
converts ⊢ Prob where ? C ∈ Prob into
⊢ ? C :: Prob.

'CLL' tactic_of !
converts ⊢ Prob where !C ∈ Prob into
⊢ [C / !C]^Prob
if check_ok(Prob).

% ILinCLL
'ILinCLL' is_an_extension_of 'CLL'.

% IMALL

'IMALL' tactic_of 'Cut'
converts A ⊢ G into
 A1 ⊢ C and
C :: A2 ⊢ G
given A equ A1 plus A2 and C is_cut_formula.

'IMALL' tactic_of '⊗L'
converts Prob ⊢ G where C ⊗ D ∈ Prob into
[(C, D) / C ⊗ D]^Prob ⊢ G.

'IMALL' tactic_of '⊸L'
converts A ⊢ G where C ⊸ D ∈ F into
A1 ⊢ C and
D :: A2 ⊢ G
given (A - C ⊸ D) equ A1 plus A2.

'IMALL' tactic_of '&L1'
converts Prob ⊢ G where C & D ∈ Prob into
[C / C & D]^Prob ⊢ G.

'IMALL' tactic_of '&L2'
converts Prob ⊢ G where C & D ∈ Prob into
[D / C & D]^Prob ⊢ G.

'IMALL' tactic_of '⊕L'
converts Prob ⊢ G where C ⊕ D ∈ Prob into
[C / C ⊕ D]^Prob ⊢ G and
[D / C ⊕ D]^Prob ⊢ G.

'IMALL' tactic_of '1L'
converts Prob ⊢ G where '1' ∈ Prob into
Prob - '1' ⊢ G.

'IMALL' tactic_of '⊸R'
converts A ⊢ C ⊸ D into
C :: A ⊢ D.

'IMALL' tactic_of '&R'
converts A ⊢ C & D into
A ⊢ C and A ⊢ D.

'IMALL' tactic_of '⊕R2'
converts A ⊢ C ⊕ D into
A ⊢ D.
```

```
'IMALL' tactic_of '⊕R1'
converts A ⊢ C ⊕ D into
A ⊢ C.

'IMALL' tactic_of '⊗R'
converts Context ⊢ C ⊗ D into
A1 ⊢ C and
A2 ⊢ D given
Context equ A1 plus A2.

% ILL

'ILL' is_an_extension_of 'IMALL'.

'ILL' tactic_of 'W!'
converts Prob ⊢ G where !C ∈ Prob into
Prob - !C ⊢ G.

'ILL' tactic_of 'D!'
converts Prob ⊢ G where !C ∈ Prob into
[C / !C]^Prob ⊢ G.

'ILL' tactic_of 'C!'
converts Prob ⊢ G where !C ∈ Prob into
!C :: Prob ⊢ G.

'ILL' tactic_of '!R'
converts A ⊢ !G into
A ⊢ G
if all_banged(A).

% ILinILL

'ILinILL' is_an_extension_of 'ILL'.

check_ok(Formulas) :-
remove_formula(!(_), Formulas, OtherFormulas),
queried(OtherFormulas).

queried([]).
queried([? _ | Rest]):-
      queried(Rest).

all_banged([]).
all_banged([! _ | Rest]):-
      all_banged(Rest).
```

```
% MALL

'MALL' axiom_of 'Identity' :
⊢ [A ⊥, A] is_axiomatic_if A is_an_atom.

'MALL' axiom_of 'Identity' :
⊢ [A, A ⊥] is_axiomatic_if A is_an_atom.

'MALL' axiom_of '1 axiom' :
⊢ ['1'] is_axiomatic.

'MALL' axiom_of 'T axiom' :
⊢ G is_axiomatic_if 'T' ∈ G.

% IMALL

'IMALL' axiom_of 'T axiom' :
A ⊢ 'T' is_axiomatic.

'IMALL' axiom_of 'Identity' :
[G] ⊢ G is_axiomatic_if G is_an_atom.

'IMALL' axiom_of '0 axiom' :
F ⊢ G is_axiomatic_if '0' ∈ F.

'IMALL' axiom_of '1 axiom' :
[] ⊢ '1' is_axiomatic.
```

INTERPRETER

```
get_axiom(Logic, Axiom, Problem, Condition) :-
      Logic axiom_of Axiom : Problem is_axiomatic_if Condition.

get_axiom(Logic, Axiom, Problem, true) :-
      Logic axiom_of Axiom : Problem is_axiomatic.

get_axiom(Logic, Axiom, Problem, Condition) :-
      Logic1 axiom_of Axiom : Problem is_axiomatic_if Condition,
      Logic is_an_extension_of Logic1.

trivial(Logic, Problem, Name) :-
      get_axiom(Logic, Name, Problem, Condition),
      Condition, !.

A is_axiomatic          :-      atom(A).
A ∈ B                   :-      on(A, B).


A is_an_atom :- atom(A).

% interpreter for tactics

apply(Stop, _, _) :-
        is_stop(S),
        S=Stop,
        abandon_problem.

apply(Tactic, Problem, NewProblems) :-
        current_logic(Logic),
        get_tactic(Logic, Tactic, Problem, Condition, Problems, _),
        choose_formula_satisfies(Condition), !,
        interpret(Problems, NewProblems), !.

apply(Axiom, Problem, []) :-
        axioms_as_tactics_set,
        current_logic(Logic),
        Logic axiom_of Axiom : Problem is_axiomatic_if Conditions,
        call(Conditions), !.

interpret(A and B, Problems) :-
        evaluate_sequent(A, Problem1),
        evaluate_sequent(B, Problem2),
        append([Problem1], [Problem2], Problems).
interpret(A and B given List equ List1 plus List2 and C is_cut_formula, [Problem1, Problem2])
:-
        evaluate_list(List, NList),
        evaluate_sequent(A, Problem1),
        evaluate_sequent(B, Problem2),
        perform_cut(NList, Problem1, Problem2, List1, List2, C).
```

```
interpret(A and B given List equ List1 plus List2, [Problem1, Problem2]) :-
        evaluate_list(List, NList),
        evaluate_sequent(A, Problem1),
        evaluate_sequent(B, Problem2),
        split_cont(NList, Problem1, Problem2, List1, List2).
interpret(Problem, [NewProblem]) :-
        evaluate_sequent(Problem, NewProblem).

evaluate_sequent(⊢A, ⊢NA) :- !,
        evaluate_list(A, NA).
evaluate_sequent(A⊢G, NA⊢G) :-
        evaluate_list(A, NA).

evaluate_list(A, A) :- var(A), !.          % Catches variables and leaves them uninstanciated
evaluate_list([(A, B)/C]^List, Result) :- !,
        evaluate_list(List, NewList),
        replace_formula(C, [A, B], NewList, Result).
evaluate_list([A/C]^List, Result) :- !,
        evaluate_list(List, NewList),
        replace_formula(C, [A], NewList, Result).
evaluate_list(List-Formula, Result) :- !,
        evaluate_list(List, NewList),
        remove_formula(Formula, NewList, Result).
evaluate_list(A::L, [A | NL]) :- !,
        evaluate_list(L, NL).
evaluate_list(A, A).

replace_formula(_, _, [], []).
replace_formula(Formula, NewFormulas, [Formula | RestList], NewList) :- !,
        append(NewFormulas, RestList, NewList).
replace_formula(Formula, NewFormulas, [First | Rest], [First | NewRest]) :-
        replace_formula(Formula, NewFormulas, Rest, NewRest).

remove_formula(Formula, Prob, NProb) :-
        one(remove(Formula, Prob, NProb)).


choose_formula_satisfies((always)).
choose_formula_satisfies(∊(Template, Formulae)) :-
        find_suitable(Template, Formulae, Suitable_Formulae),
        (Suitable_Formulae = [] -> !, fail          % Should never happen
        ;Suitable_Formulae = [Template]             % Instanciates Template to the one suitable
formula
        ;select_one(Suitable_Formulae, Template)), !.   % Instanciates Template to the selected
suitable formula

find_suitable(Template, Formulae, Suitable) :-
        setof(Template, (on(Template, Formulae)), Suitable).

get_tactics(Logic, Problem, Applicable, All) :-
      all_tactics(Logic, All),
      applicable_tactics(Logic, Problem, Applicable).

all_tactics(Logic, Tactics) :-
      bagof(Tactic, (get_one_tactic(Logic, Tactic)), Tactics), !.
all_tactics(_, []).
```

```
get_one_tactic(Logic, Tactic) :-
      get_tactic(Logic, Tactic, _, _, _, _).
get_one_tactic(Logic, Axiom) :-
      axioms_as_tactics_set,  % Axioms may be applied as tactics
      get_axiom(Logic, Axiom, _, _).

applicable_tactics(Logic, Problem, Tactics) :-
      bagof(Tactic, (get_one_applicable_tactic(Logic, Problem, Tactic)), Tactics), !.
applicable_tactics(_, _, []).

get_one_applicable_tactic(Logic, Prob, Tact) :-
      get_tactic(Logic, Tactic, Prob, Condition, _, SideCondition),
      Condition,
      Side_Condition.

get_one_applicable_tactic(Logic, Problem, Name) :-
      axioms_as_tactics_set,
      get_axiom(Logic, Name, Problem, Conditions),
      Conditions.

A∈B :- on(A, B).
(always).

%       Interpreter for menu

get_all_logics(Logics) :-
        get_all_logics(Logics, []).

get_all_logics(A, Logics) :-
        get_tactic(Logic , _Tact , _Prob , _G , _ , _Cond2),
        not(on(Logic, Logics)),
        append(Logics, [Logic], W),
        get_all_logics(A, W).
get_all_logics(A, A).


get_tactic(Logic, Tactic, Problem, true, Sprobs , SideCondition) :-
      Logic tactic_of Tactic converts Problem into Sprobs SideCondition.

get_tactic(Logic, Tactic, Problem, Condition, NewProbs, true) :-
      Logic tactic_of Tactic converts Problem where Condition into NewProbs.

get_tactic(Logic, Tactic, Problem, true, Sprobs , true) :-
      Logic tactic_of Tactic converts Problem Sprobs NewProbs.

get_tactic(Logic, Tactic, Problem, Condition, Sprobs , SideCondition) :-
      Logic is_an_extension_of Logic1,
      get_tactic(Logic1, Tactic, Problem, Condition, Sprobs , SideCondition).
```

THEOREM PROVER

```
prove_ill(A ⊢ G, MaxDepth) :-
      check_and_sort_and_prove_ill(A, [], G, U, MaxDepth),
      all_banged(U).

all_banged(U) :- map(banged_formula, U).
banged_formula( ! _).

check_and_prove_ill(G, A, U, MaxDepth) :-
      remove(G, A, U).

check_and_prove_ill('1', A, A, MaxDepth).

check_and_prove_ill(T, A, U, MaxDepth) :-
      (var(U) -> generate_submulti_set(A, U)
      ; submulti_set(A, U))

check_and_prove_ill(G, A, U, MaxDepth) :-
      prove_ill(G, A, U, MaxDepth).

prove_ill(G, A, U) :-
      remove_pair(A, U).

prove_ill(C&D, A, U, MaxDepth) :- !,
      check_and_prove_ill(C, A, U, MaxDepth),
      check_and_prove_ill(D, A, U, MaxDepth).

prove_ill(C⊸D, A, U, MaxDepth) :- !,
      check_and_sort_and_prove_ill([C], A, D, U, MaxDepth),
      submulti_set(A, U).

prove_ill(!G, A, U, MaxDepth) :- !,
      get_banged_context(A, Banged, UnBanged),
      check_and_prove_ill(G, Banged, U1, MaxDepth),
      append(A, UnBanged, U).

prove_ill(G, A, U, MaxDepth) :-
      remove('1', A, NA), !,
      prove_ill(NA, G, U, MaxDepth),
      submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
      remove(C⊗D, A, NA),
      sort_and_prove_ill([C, D], NA, G, U, MaxDepth),
      submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
      remove(C⊕D, A, NA), !,
      sort_and_prove_ill([C], NA, G, U, MaxDepth),
      sort_and_prove_ill([D], NA, G, U, MaxDepth),
      submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
      remove_banged(!C, _, A, NA, MaxDepth),
      sort_and_prove_ill([C], NA, G, U, MaxDepth),
      submulti_set(NA, U).
```

```
prove_ill(G, A, U, MaxDepth) :-
        remove(C&D, A, NA),
        (sort_and_prove_ill([C], NA, G, U, MaxDepth)
        ;sort_and_prove_ill([D], NA, G, U, MaxDepth)
        ),
        submulti_set(NA, U).

prove_ill(C⊕D, A, U, MaxDepth) :-
        ( check_and_prove_ill(C, A, U, MaxDepth)
        ; check_and_prove_ill(D, A, U, MaxDepth)).

prove_ill(C⊗D, A, U, MaxDepth) :-
        check_and_prove_ill(C, A, U1, MaxDepth),
        check_and_prove_ill(D, U1, U, MaxDepth).

prove_ill(G, A, U, MaxDepth) :-
        remove(C-∘D, A, NA),
        check_and_prove_ill(C, NA, U1, MaxDepth),
        check_and_prove_ill(G, [D | U1], U, MaxDepth),
        submulti_set(NA, U).

prove_ill(G, A, U, MaxDepth) :-
        remove_banged(!C, D, A, NA, MaxDepth),
        D ≥ 0, !,
        D1 is D - 1,
        sort_and_prove_ill([!(C, 0), !(C, D1)], NA, G, U, MaxDepth),
        submulti_set(NA, U).

sort_and_prove_ill(F, A, G, U, MaxDepth) :-
        remove('0', F, NF),
         (var(U) -> generate_submulti_set(NF, U)
        ; submulti_set(NF, U))

sort_and_prove_ill(F, A, G, U, MaxDepth) :-
        remove(G, F, NF),
        append(A, NF, U).
sort_and_prove_ill(F, A, G, U, MaxDepth) :-
        append(F, A, NA),
        prove_ill(G, NA, U, MaxDepth).

check_and_sort_and_prove_ill(F, A, G, U, MaxDepth) :-
        remove('0', F, NF),
        (var(U) -> generate_submulti_set(NF, U)
        ; submulti_set(NF, U))
check_and_sort_and_prove_ill(F, A, G, U, MaxDepth) :-
        append(F, A, NA),
        check_and_prove_ill(G, NA, U, MaxDepth).

remove_banged(!F, MD, A, U, MD) :-
        remove(!F, A, U).
remove_banged(!F, D, A, U, _) :-
        remove(!(F, D), A, U).

submulti_set(_, []).
submulti_set(A, [F | R]) :-
        remove(F, A, RA),
        submulti_set(RA, R).
```

```
generate_submulti_set([], []).
generate_submulti_set([F | R], [F | R1] ) :-
      generate_submulti_set(R, R1).
generate_submulti_set([F | R], R1) :-
      generate_submulti_set(R, R1).

get_banged_context([], [], []).
get_banged_context([!F | R], [!F | B], UB) :- !,
      get_banged_context(R, B, UB).
get_banged_context([F | R], B, [F | UB]) :- !,
      get_banged_context(R, B, UB).
```

```
remove_pair([F | R], U) :-
      negate(F, NF),
      remove(NF, R, U), !.
remove_pair([F | R], [F | U]) :-
      remove_pair(R, U).

prove_mall(⊢ A) :-
      map(negate, A, A1)
      map(transform_all, A1, NA),
      prove_imall3(NA ⊢ ⊥).

transform_all((A ⊗ B)⊥, C) :- transform_all(A⊥ ⅋ B⊥, C),!.
transform_all((A & B) ⊥ , C) :- transform_all(A⊥ & B⊥, C),!.
transform_all((A ⊕ B) ⊥ , C) :- transform_all(A⊥ ⅋ B⊥, C),!.
transform_all((A ⅋ B) ⊥ , C) :- transform_all(A⊥ ⊕ B⊥, C),!.
transform_all((A ⊸ B) ⊥ , C) :- transform_all(A ⊕ B⊥, C),!.
transform_all(A ⅋ B, C) :-      % ⅋ not a connective of imall
      transform_all((A⊥ ⊸ B)&(B⊥ ⊸ A), C), !.
transform_all(Op(A, B) , Op(NA, NB)) :- !,
      transform_all(A, NA),
      transform_all(B, NB).

transform_all( '0'⊥, 'T').
transform_all( 'T'⊥, '0').
transform_all( '1'⊥, '⊥').
transform_all( '⊥'⊥, '1').
transform_all((A⊥)⊥, C) :- transform_all(A, C).
transform_all(A ⊥ , NA ⊥) :- !,
      transform_all(A, NA).

transform_all( A, A) :- atomic(A).



 '<INTERRUPT>'(Callterm) :-
        theorem_proving(on),
        ask_interupt_theorem_prover, !,
        remember(interupt, on),
        continue(Callterm).

 '<INTERRUPT>'(Callterm) :-
        theorem_proving(on), !,
        continue(Callterm).

 '<INTERRUPT>'(Callterm) :-
        proof_editing(on),
        ask_interupt_proof_editing, !,
        abandon_problem.
'<INTERRUPT>'(Callterm) :-
        proof_editing(on), !,
        continue(Callterm).
'<INTERRUPT>'(Callterm) :-
        ( ask_interrupt_current_process
        ; !, continue(Callterm)
        ),
        abort.
```

```
continue(Call) :-
        def(Call), !,
        Call.
continue(Call) :-
        $(Call).

ask_interrupt_current_process :-
        centered_yes_no(['Interrupt the current process?']).

ask_interupt_proof_editing :-
        centered_yes_no(['Abandon Current Problem?']).

theorem_prover('MALL', prove_mall).
theorem_prover('ILL', prove_ill).
theorem_prover('IMALL', prove_ill).
theorem_prover('ILinILL', prove_ill).
theorem_prover('IILinIMALL', prove_ill).

maybe_check(Logic, Problems) :-
        check_theorem_prover(on),
        search_depth(Depth),
        theorem_prover(Logic, Op),
        remember_theorem_proving_on,
        remember(interupt, off),
        (on(Problem, Problems),
        not(Op(Depth, Problem)),
        recall(interupt, off), !,
        inform_not_valid(Logic, Problem, Depth),
        remember_theorem_proving_off
        ;true,
        remember_theorem_proving_off), !.
maybe_check(_, _) :- remember_theorem_proving_off, !.


inform_not_valid(Logic, Problem, Depth) :-
        check_decidable(Problem, Status),
        not_valid_warning(Problem, Logic, Depth, Status).

check_decidable(Prob, no) :- modal_problem(Prob), !.
check_decidable(Prob, yes).

modal_problem(A⊢B) :-
        modal_list(A)
        ;modal_formula(B).
modal_problem(⊢B) :-
        modal_list(B).

modal_list(A) :-
        on(F, A),
        modal_formula(F).

modal_formula(?_).
modal_formula(!_).
modal_formula(Op(A, B)) :-
        modal_formula(A)
        ;modal_formula(B).
modal_formula(Op(A)) :-
        modal_formula(A).
```

PROOF EDITOR

```
solve(Window, Logic, Problem) :- solve(Window, Logic, 2, Problem).
                                                  % Initial Depth in proof tree is 1

solve(Window, Logic, _Depth, Problem) :-
        trivial(Logic, Problem, Name),
        !,
        write_terminator(Window, Name),
        find_next_problem(Window).

solve(Window, Logic, Depth, Problem) :-
        repeat,
        get_applicable_tactic(Problem, Logic, Tactic),
        (is_backtrack(Tactic) -> !, fail  % fail and backtrack if the selected tactic is
'Backtrack'
        ;true
        ),
        apply(Tactic, Problem, Subproblems),  % Generate the new subproblems
        transform_problems(Logic, Subproblems, NewSubproblems),
        maybe_check(Logic, NewSubproblems), % If the theorem prover is turned on, check the problems
        proof_write(Window, Tactic, Depth, NewSubproblems),  % Write out the new problems
        NewDepth is Depth +1,
        solve_problems(Window, Logic, NewDepth, NewSubproblems).  % Solve the new problems

solve_problems(_Window, _Logic, _Depth, []) :- !.
solve_problems(Window, Logic, Depth, [Problem | OtherProblems]) :-
        solve(Window, Logic, Depth, Problem),
        solve_problems(Window, Logic, Depth, OtherProblems).

start_problem(Problem) :-
        remember_proof_editing_on,
        proof_window_to_front,
        clear_proof_window,
        disable_menus,
        lock_proof_window,
        current_logic(Logic),
        proof_window(Window),
        initial_text(Window, Logic, Problem),
        initial_transform(Logic, Problem, Problem1),
        transform_problems(Logic, [Problem1], [Problem2]),
        write_sequents(Window, [Problem2], 1),
        maybe_check(Logic, [Problem2]),
        solve(Window, Logic, Problem2),
        congratulate_user(Window),
        reenable_menus,
        clear_database,
        unlock_proof_window,
        remember_proof_editing_off.
start_problem(_) :-
        abandon_problem.
```

```
abandon_problem :-
        reenable_menus,
        clear_database,
        unlock_proof_window,
        remember_proof_editing_off,
        write_problem_abandonned,
        abort.

initial_text(Window, Logic, Problem) :-
        logic_text(Logic, T1, T2, T3),
        write_b(Window, T1),
        nl_b(Window),
        write_b(Window, T2),
        nl_b(Window),
        sequent_write(Window, Problem, 0),
        nl_b(Window),
        nl_b(Window),
        write_b(Window, T3),
        nl_b(Window).
get_applicable_tactic(Problem, Logic, Tactic) :-
        get_tactics(Logic, Problem, Applicable_tactics, All_tactics),
        is_backtrack(Backtrack),        % Load the text 'Backtrack' from resources
        is_stop(Stop),  % Load the text 'Stop' from resources
        append(All_tactics, [Backtrack, Stop], All_tactics2),
        append(Applicable_tactics, [Backtrack, Stop], Applicable_tactics2),
        (applicable_tactics_wanted -> Tactics = Applicable_tactics2     % Display only
applicable tactics
        ;otherwise -> Tactics = All_tactics2    % Display all tactics
        ),
        choose_one_dialog(Tactics, Applicable_tactics2, Tactic, % Allow the user to choose a tactic
        applicable_tactic(Tactic, Applicable_tactics2)  % which is applicable.
        ), !.

applicable_tactic(Tactic, Possible_Tactics) :-
        not(on(Tactic, Possible_Tactics)),      % Tactic not applicable
        that_tactic_isnt_applicable_message(Tactic),  % Inform the user
        !,
        fail.   % and fail
applicable_tactic(_, _).                % Otherwise succeed.
```

PARSER/UNPARSER

```
formula(A)        --> formula(A, _, 1200) .

formula(A,M,N) --> formula0(A1),
                   more_formulas(A1, 0, N, A, M) .

formula(A,M,N)  --> prefix_op( Op, N1), { N1 ≤ N },
                   formula(B2, _, N1),
                   {OpB2 =.. [ Op, B2] },
                   more_formulas( OpB2, N1, N, A, M ).

formula0(⟦ A ⟧)       --> lp(z), formula(A, _, 1200), rp(z).  % Tom
formula0(A)       --> lp(Kind), formula(A, _, 1200), rp(Kind).

formula0(V)       --> atom_formula(V).

atom_formula( '⋀' )             -->  "⋀" | "f".
atom_formula( '⊥' )             -->  "⊥".
atom_formula( '1' )             -->  "1".
atom_formula( '0' )             -->  "0".
atom_formula( 'Ω' )             -->  "Ω".
atom_formula( 'T' )             -->  "T".
atom_formula( 'k' )             -->  "k".

atom_formula( P )  --> [C],  { C  ≥ 65,         % 65 = ASCII for "A"
                              C  ≤ 90,        % 90 = ASCII for "Z"
                              name(P,[C]) }.

more_formulas(A1, W, N, A, M) -->
   postfix_op( Op, U), { U =   P, W ≤ P, P ≤ N},
   more_formulas(Op(A1), P, N, A, M).


more_formulas(A1, W, N, A, M) -->
   infix_op( Op, U, K), { U = L * R → P, W ≤ L, P ≤ N},
   formula_right(K, Op, U, A1, N, A2, M2),
   more_formulas(A2, M2, N, A, M).

more_formulas(A, W, _, A, W) --> "".


formula_right( yfx, Op, L * R → P, A1, N, A, M)
   --> !,
   formula(A2, _, R),
   more_formulas( Op(A1, A2), P, N, A, M).
formula_right( K, Op , L * R → P, A1, N, Op(A1, A2), P)
  -->
   formula(A2, _, R).

lp(round)              --> "(".
lp(square)             --> "[".
rp(round)              --> ")".
rp(square)             --> "]".
lp(z)          --> "⟦".
rp(z)            --> "⟧".
```

```
initial_syntax('ILinCLL', 'IL').
initial_syntax('ILinILL', 'IL').
initial_syntax('IILinIMALL', 'IIL').

parse(Bytes, FORMULA) :-
   current_logic( Logic ) ,
   accept(Logic, Bytes, OKBytes), !,
   phrase(formula(FORMULA), OKBytes).

parse_initial(Bytes, FORMULA) :-
   current_logic( Logic ) ,
         (initial_syntax(Logic, Initial_Logic), ! ; Initial_Logic = Logic),
   accept(Initial_Logic, Bytes, OKBytes), !,
   phrase(formula(FORMULA), OKBytes).

my_parse(_dlg, _btn, Bytes, FORMULA) :-
  my_parse( Bytes, FORMULA).

my_parse(Bytes, FORMULA) :-
   parse(Bytes, FORMULA) , ! ;
   name(Atom, Bytes),
   (Atom = '' -> missing_formula_message
             ;  quote( Atom , QA ) ,
                not_valid_formula_message(QA)
         ),
         fail .

my_parse_initial(_dlg, _btn, Bytes, FORMULA) :-
  my_parse( Bytes, FORMULA).

my_parse_initial(Bytes, FORMULA) :-
   parse_initial(Bytes, FORMULA) , ! ;
   name(Atom, Bytes),
   (Atom = '' -> missing_formula_message
             ;  quote( Atom , QA ) ,
                not_valid_formula_message(QA)
         ),
         fail .

bytes([])  -->  "".
bytes([H | T])  --> non_commas(H), ",",  !, bytes(T).
bytes([H])   --> non_commas(H).

non_commas([B|Bs]) --> [B], { B \= 44 }, non_commas(Bs).
non_commas([])  --> "".

parse_list( [] , [] ) .
parse_list( [ F | R ] , [ F1 | R1 ] ) :-
  name(F,F2),
  parse( F2 , F1 ) ,
  parse_list( R , R1 ).
```

```
check_and_parse(AssBytes , GoalBytes , Assums , Goal ) :-
  phrase( bytes( AssBytesList ) , AssBytes ) , ! ,
  map( my_parse_initial , AssBytesList , Assums ) , ! ,
  current_logic( Logic ) , sequent_type( Logic, Type ) ,
  ((Type = list ⊢ list)-> phrase(bytes(GoalBytesList),GoalBytes) , ! ,
                                          map( my_parse_initial , GoalBytesList , Goal) , !
    ; Type = list ⊢ formula                          -> my_parse_initial( GoalBytes , Goal ) ) .


parse_problem( Bytes , Assums , Goal ) :-
  (phrase( turn( AssBytesList , GoalBytesList ) ,Bytes ) ; not_a_sequent_message,fail), !,
  map( my_parse_initial , AssBytesList , Assums ) , ! ,
  current_logic( Logic ) , sequent_type( Logic, Type ) ,
  ((Type = list ⊢ list)-> map( my_parse_initial , GoalBytesList , Goal) , !
    ; (Type = list ⊢ formula) -> GoalBytesList = [ GoalBytes ] ,my_parse_initial( GoalBytes ,
Goal ) ) .

parse_problem_same_again( Bytes , Assums , Goal ) :-
  phrase( turn( AssBytesList , GoalBytesList ) ,Bytes ) ,
  map( parse_initial , AssBytesList , Assums ) ,
  current_logic( Logic ) , sequent_type( Logic, Type ) ,
  ((Type = list ⊢ list)-> map( parse_initial , GoalBytesList , Goal)
    ; (Type = list ⊢ formula) -> GoalBytesList = [ GoalBytes ] ,parse_initial( GoalBytes , Goal )
) .

parse_problem_same_again(Bytes, _, _) :-
        current_logic(Logic),
        name(Bytes_text, Bytes),
        centered_message([Bytes_text, 'is not a valid sequent in', Logic]),
        fail.


turn( A , B ) --> bytes1( A ) , "⊢" , bytes1( B ) .

bytes1([])  -->  "".
bytes1([H | T])  --> non_commas1(H), ",",  !, bytes1(T).
bytes1([H])   --> non_commas1(H).

non_commas1([B|Bs]) --> [B], { B \= 44 , B \= 230 }, non_commas1(Bs).
non_commas1([])  --> "".



unparse( X , Atom ) :-
 phrase( unparse2( X ) , S ) ,
 name( Term , S ) ,
 ( number(Term) -> pname( Term, Atom) ; otherwise -> Atom = Term ) .

unparse2( all( X , PX ) )    --> ! , unparse1( all( X , PX ) , 1200 ) .
unparse2( some( X , PX ) ) --> ! , unparse1( some( X , PX ) , 1200 ) .
unparse2( Op( A , B ) )      --> ! , { infix( C , Op , M1 * M2 → M , K ) } ,
                                   unparse1( A , M1 ) , "" , char( Op ) , "" , unparse1( B , M2 )
.
unparse2( A )                --> unparse1( A , 1200 ) .
```

```
unparse1(  Pred  , _ )     --> {atom(Pred) }, ! , char( Pred ) .

unparse1( X , 0 ) --> ! , unparse1( X , 1200 ) .

unparse1( Op(A) , N ) --> {Op = '⟦'} , unparse1( A , N ) , "⟧" .
unparse1( Op(A) , N ) --> "⟦" , unparse1( A , N ) , {Op = '⟧'}.

unparse1( Op( P ) , N ) --> { prefix( C , Op , M1) } ,
  ( { M1 ≤ N } , ! , [ C ] , " (", unparse1( P , M1 ), ")"
  | unparse1( Op( P ) , 0 )
  ).

unparse1( Op( P ) , N ) --> { postfix( C , Op , M1) } ,
  ( { M1 ≤ N } , ! , unparse1( P , M1 )
  | unparse1( Op( P ) , 0 ) ) , [ C ] .

unparse1( Op( P ,Q ) , N ) -->
  { infix( B , Op , M1 * M2 → M , K)  } ,
  (  "(" , { M ≤ N } , ! , unparse1( P , M1 ) , [ B ] , unparse1( Q , M2 ) , ")"
  | unparse1( Op( P , Q ) , 0 )
  ).

char( Op )          -->  { charof( Op , B ) } , [ B ] .

unparse_list( Formulas , UnparsedFormulas ) :-
  map(unparse , Formulas , UnparsedFormulas ) .



unparse_sequent(  ⊢ B , R ) :-
  unparse_list( B , UB0 ) , comarise( UB0 , UB ) , concat( [ '⊢ ' | UB ] , R ) .

unparse_sequent( A ⊢ B , R ) :-
  lst( B ) , ! ,
   unparse_list( A , UA0 ) , comarise( UA0 , UA),concat( UA , UA1 ) ,
   unparse_list( B , UB0 ) , comarise( UB0 , UB),concat( UB , UB1 ) ,
  concat( [ UA1 , ' ⊢ ' , UB1 ] , R ) .

unparse_sequent( A ⊢ B , R ) :-
  unparse_list( A , UA0 ) , comarise( UA0 , UA),concat( UA , UA1 ) ,
        unparse( B , NB ) ,
  concat( [ UA1 , ' ⊢ ' , NB ] , R ) .

comarise( [] , [] ) .
comarise( [ A ] , [ A ] ) :- ! .
comarise( [ A | R ] , [ NA | NR ] ) :- concat( A , ', ' , NA ) , comarise( R , NR ) .
```

```
prefix_op( Op , Prec)   -->  [C], {prefix(C, Op,Prec)}.
infix_op( Op , Prec, K) --> [C], { infix(C, Op, Prec, K)}.
postfix_op( Op , Prec) --> [C], { postfix(C, Op, Prec)}.

prefix(241,  ¬ , 750 ) :- !. %%  "~"
prefix( 45,  ¬ , 750 ).      %% "-"
prefix(209,  ¬ , 750 ).      %% "—"
prefix(208,  ¬ , 750 ).     %%  "-"
prefix(194,  ¬ , 750 ).      %% "¬"

prefix(33,  ! , 750 ).     %%  !  "of course"
prefix(63,  ? , 750 ).        %% ?  "why not"


postfix( 248, ⊥, 750 ).    %% involution
postfix( 45, -, 750 ).
postfix( 43, +, 750 ).

infix(235, ∨, 1000 * 999    → 1000, yfx).
infix( 38, &, 950  * 949     →  950,  yfx).
infix( 234, ↔, 1049 * 1049   → 1050,   xfx).
infix(231,  '→',  1049 * 1050   → 1050,   xfy) :- !.  % "→"
infix(250,  '→',  1049 * 1050   → 1050,   xfy).   % "⊃"
infix( 225, '∍',  1049 * 1050   → 1050,   xfy).

infix( 224, ⅋, 950  * 949     →   950,  yfx).
infix( 187, ⊗, 950  * 949      →   950, yfx).
infix( 188, ⊕, 950  * 949     →   950,  yfx).
infix( 180,  '⊸',  1049 * 1050  → 1050,   xfy).  % "linear implication"

% accept( String, OKString )
% converts a String to a list of acceptable characters


accept( _, [], [] ).
accept(Logic, [ C | T1 ], [ C | T2] ) :-
      symbol_for([C], Logic ), !,
     accept(Logic, T1, T2).
accept( Logic,  [C | T1], T2) :-
     ignorable(C), !,
     accept(Logic, T1, T2).

ignorable( 32 ).                 % Space
ignorable( 202 ).                % Option space
ignorable( 9).                   % Tab
ignorable( 227).                 % Shift Option W, i.e. tiny space
ignorable( 13).                  % Return

symbol_for([A], _) :-
       A ≥ 65,
       A ≤ 90.
symbol_for("(", _).
symbol_for(")", _).
symbol_for("[", _).
symbol_for("]", _).      % Belong to all the logics
symbol_for("⊗", 'IMALL').
symbol_for("⊕", 'IMALL').
symbol_for("&", 'IMALL').
symbol_for("⊸", 'IMALL').
symbol_for("1", 'IMALL').
symbol_for("0", 'IMALL').
symbol_for("⊤", 'IMALL').

'ILL' is_a_syntactic_extension_of 'IMALL'.
symbol_for("!", 'ILL').

'ILinILL' is_a_syntactic_extension_of 'ILL'
'ILinILL' is_a_syntactic_extension_of 'IL'.
symbol_for("⟦", 'ILinILL').
symbol_for("⟧", 'ILinILL').

symbol_for("&", 'IL').
symbol_for("→", 'IL').
symbol_for("∧", 'IL').
symbol_for("∨", 'IL').
symbol_for("~", 'IL').

symbol_for("⅋", 'MALL').
'MALL' is_a_syntactic_extension_of 'IMALL'.

'CLL' is_a_syntactic_extension_of 'MALL'.
symbol_for("!", 'CLL').
symbol_for("?", 'CLL').

'ILinCLL' is_a_syntactic_extension_of 'CLL'.
'ILinCLL' is_a_syntactic_extension_of 'IL'.
symbol_for("⟦", 'ILinCLL').
symbol_for("⟧", 'ILinCLL').

symbol_for(X, Logic1) :-
      Logic1 is_a_syntactic_extension_of Logic,
      symbol_for(X, Logic).
```

USER INTERFACE

```
write_probs3(Dialog, 3, Logic, Problem1, Problem2, All_items, Parsed_Selected_Items,
Parsed_Rest_Items, D, Cut) :- !,
        parse(Cut, D),
        getditem(Dialog, 8, _, _, _, Selected_Items, _),
        parse_list(Selected_Items, Parsed_Selected_Items),
        parse_list(All_items, All_Parsed_Items),
        transform_problems(Logic, [Problem1], [NewProblem1]),
        unparse_sequent(NewProblem1, UnparsedProblem1),
        setditem(Dialog, 5, UnparsedProblem1), !,
        split_list(All_Parsed_Items, Parsed_Selected_Items, Parsed_Rest_Items),
        transform_problems(Logic, [Problem2], [NewProblem2]),
        unparse_sequent(NewProblem2, UnparsedProblem2),
        setditem(Dialog, 7, UnparsedProblem2), !,
        fail.
write_probs3(Dialog, 1, Logic, Problem1, Problem2, All_items, Parsed_Selected_Items,
Parsed_Rest_Items, D, Cut):- !,
        parse(Cut, D),
        getditem(Dialog, 8, _, _, _, Selected_Items, _),
        parse_list(Selected_Items, Parsed_Selected_Items),
        parse_list(All_items, All_Parsed_Items),
        transform_problems(Logic, [Problem1], [NewProblem1]),
        unparse_sequent(NewProblem1, UnparsedProblem1),
        split_list(All_Parsed_Items, Parsed_Selected_Items, Parsed_Rest_Items),
        transform_problems(Logic, [Problem2], [NewProblem2]),
        unparse_sequent(NewProblem2, UnparsedProblem2), !.

get_probs_initial3(Logic, List, Prob1, Prob2, Tprob1, Tprob2) :-
        toground(Prob1, Prob11),
        (tohollow(Prob11, Prob12, ['_1', '_2'], ['A', []]);tohollow(Prob11, Prob12, ['_1',
                                                                     '_2'], [[], 'A'])),
        toground(Prob2, Prob21),
        tohollow(Prob21, Prob22, ['_1', '_2'], ['A', List]),
        transform_problems(Logic, [Prob12], [NProb12]),
        transform_problems(Logic, [Prob22], [NProb22]),
        unparse_sequent(NProb12, Tprob1),
        unparse_sequent(NProb22, Tprob2).

get_probs_initial2(Logic, List, Prob1, Prob2, Tprob1, Tprob2, D) :-
        copy_term(Prob1, Prob12),
        varsin(Prob12, [A1, B1]),
        (varsin(Prob1, [Z, _]), Z == D ->  A1 = 'A', B1 = []
        ; varsin(Prob1, [_, Z]), Z == D -> B1 = 'A', A1 = []
        ),
        copy_term(Prob2, Prob22),
        varsin(Prob22, [A2, B2]),
        (varsin(Prob2, [Z, _]), Z == D  ->  A2 - 'A', B2 = List
        ; varsin(Prob2, [_, Z]), Z == D  -> B2 = 'A', A2 = List
        ),
        transform_problems(Logic, [Prob12], [NProb12]),
        transform_problems(Logic, [Prob22], [NProb22]),
        unparse_sequent(NProb12, Tprob1),
        unparse_sequent(NProb22, Tprob2).
```

```
get_window_name(Default_window_name, New_window_name, Goal) :-
        centered_modal_dialog(100, 190, [       button(70, 10, 20, 60, ok.res),
                                                button(70, 120, 20, 60, cancel.res),
                                                text(10, 10, 16, 170,
                                                        enter_new_window_name.res),
                                                edit(30, 10, 16, 170, Default_window_name,
                                                                New_window_name)
                                      ],
                                      _Button,
                                      Goal).

enter_new_window_name(Old_window_name, New_window_name, Goal) :-
        quote(Old_window_name, Old_Text),
        centered_modal_dialog(100, 300, [       button(70, 20, 20, 120, ok.res),
                                                button(70, 160, 20, 120, cancel.res),
                                                text(10, 10, 40, 280,
                                                wseq([change.res, Old_Text, to.res])),
                                                edit(40, 10, 16, 280, Old_window_name,
                                                        New_window_name)
                                      ],
                                      _Button,
                                      Goal).

ask_really_kill_window(Window) :-
        quote(Window, Window_text),
        centered_yes_no([really_kill_window.res, Window_text, question_mark.res]).

unacceptabe_character_message(Character, Ascii) :-
        centered_message(['UNACCEPTABLE character.',
              '~MIt is', Character, 'and has ASCII code =', Ascii, '.']).

ask_interupt_theorem_prover :-
        centered_yes_no(['Interrupt the theorem prover?']).

that_window_already_exists_message(Window) :-
        quote(Window, Window_text),
        centered_message([the_window.res, Window_text, already_exists.res]).

not_a_sequent_message :-
        centered_message(['The selected text is not a sequent!']).

that_doesnt_work_on_dialogs_message :-
        centered_message([that_doesnt_work_on_dialogs.res]).

that_tactic_isnt_applicable_message(Tactic) :-
        centered_message([the_tactic_of.res, Tactic, is_not_applicable.res]).

loading_file_banner(File, Goal) :-
        maybe_get_text(loading_the_file.res, Loading_the_file),
        quote(File, File_name),
        banner(Goal, [Loading_the_file, File_name]).

quote(Atom, NewAtom) :-
        concat(['"', Atom, '"'], NewAtom).

ask_save_changes(Window) :-
        quote(Window, Window_text),
        centered_yes_no([save_changes_to.res,Window_text, question_mark.res]).
```

```
ask_really_quit :-
        centered_yes_no([do_you_really_want_to_quit.res]).

not_valid_warning(Problem, Logic, Depth, Status) :-
        (Status=no -> T = may_not_be.res
        ;Status=yes -> T = is_not.res
        ),
        unparse_sequent(Problem, Problem_text),
        pname(Depth, Depth_text),
        maybe_get_text(provable_in.res, Provable_in),
        maybe_get_text(level.res, Level),
        maybe_get_text(right_bracket.res, Right_bracket),
        maybe_get_text(T, T_text),
        concat([T_text, Provable_in, Logic, Level, Depth_text, Right_bracket], Warning),
        my_mdialog(235, 8, 100, 497, [
                                        button(77, 410, 20, 80, continue.res),
                                        button(77, 5, 20, 80, cancel.res),
                                        icon(22, 4, 32, 32, 0),
                                        line(1, 44, 73, 2), %vertical, width2
                                        text(1, 52, 16, 443,
                                        according_to_the_validity_checker.res),
                                        text(22, 52, 16, 443, Problem_text),
                                        text(43, 52, 32, 443, Warning)
                                  ], Btn), !.

enter_problem(Logic, A ⊢ G, ProblemBytes) :-
        sequent_type(Logic, Type),
        (Type = list ⊢ list -> Text = '(Separated by commas)', Goal_text = 'Goals:'
        ;otherwise -> Text = '', Goal_text = 'Goal:'
        ),
        last_sequent_in_dialog(OldAssumptions, OldGoals), !,
        my_dialog('Problem Entry', 71, 90, 200, 332, [button(170, 108, 20, 40, 'OK'),
                                                     button(170, 242, 20, 70, 'Cancel'),
                                                     edit(20, 110, 80, 200,
                                                     bytes(OldAssumptions),
                                                     bytes(NewAssumptions)),
                                                     edit(120, 110, 35, 200,
                                                     bytes(OldGoals),
                                                     bytes(NewGoals)),
                                                     text(20, 10, 15, 90, 'Assumptions:'),
                                                     text(40, 10, 45, 90, '(Seperated by
                                                                         commas)'),
                                                     text(140, 10, 45, 90, Text),
                                                     text(120, 10, 15, 90, Goal_text)],
                                                     _Btn,
                                                     check_and_parse(NewAssumptions,
                                                     NewGoals, A, G)),
        set_last_sequent_in_dialog(NewAssumptions, NewGoals),
        append(NewAssumptions, "⊢", Z),
        append(Z, NewGoals, ProblemBytes).
```

```
select_one(Formulae, Formula) :-
        unparse_list(Formulae, Unparsed_Formulae),
        Unparsed_Formulae=[F | _], !,
        my_dialog(select_one_dialogue.res, 40, 7, 299, 130, % Formulas
        [button(270, 17, 20, 100, ok.res),
        button(240, 17, 20, 100, cancel.res),
        text(6, 10, 15, 130, select_a_formula.res),
        menu(50, 15, 180, 100, Unparsed_Formulae, % Select a Formula
        F, Unparsed_formula)
        ],
        _Btn), !,
        get_parsed_formula(Unparsed_formula, Unparsed_Formulae, Formulae, Formula), !.

%       Because of the way that the parser and unparser work, if one parses then unparses a
        formula,
%       what one gets may not be what one originally started with (The new formula may have
                                                                          more brackets).
%       This may cause problems eg remove( (a & b), [ a & b, a ⊗ d] ) will fail.
% get_parsed formula is used to make sure that the formula returned by select one is actually
% identical to a formula in the input list.  Simply parsing the selected unparsed formula would
not necessarilyachieve this.

get_parsed_formula(UF, [UF | _], [F | _], F) :- !.
get_parsed_formula(UF, [_ | R], [_ | R1], F) :-
        get_parsed_formula(UF, R, R1, F).

choose_one_dialog( Tactics, Applicable_tactics, Tactic, Goal ) :-
Tactics = [ FirstTactic | _ ] ,          % First tactic will be initially selected
   my_dialog( tactic_choice.res , 40 , 7 , 299 , 130 ,
           [ button( 270 , 17 , 20 , 100 , ok.res ) ,
            text( 6 , 10 , 15 , 130 , select_a_tactic.res ) ,
             text( 20 , 10 , 15 , 130 , or_operation.res ) ,
            menu( 40 , 15 , 225, 100 , Tactics ,
                    FirstTactic , Tactic )
           ] ,
         _Btn , Goal ), ! .

no_text_selected_message :-
        maybe_get_text(no_text_selected.res, Text),
        centered_message([Text]).

missing_formula_message :-
 centered_message(['Missing formula in edit field!']).

not_valid_formula_message(Formula) :-
        centered_message([Formula, 'is not a valid formula!']).

reading_and_parsing_banner(Goal) :-
        banner(Goal, ['Reading and Parsing text in Front Window']).
centered_modal_dialog(Depth, Width, Format_List, Button, Goal) :-
        centred(Down_pos, In_pos, Depth, Width),
        get_dialog_resources(Format_List, New_Format_List),
        mdialog(Down_pos, In_pos, Depth, Width, New_Format_List, Button, Goal).
```

```
centered_yes_no(List) :-
        centered_modal_dialog(65, 400, [     button(10, 340, 20, 50, yes.res),
                                              button(40, 340, 20, 50, no.res),
                                              text(6, 55, 48, 275, wseq(List)),
                                              icon(17, 14, 32, 32, 2)
                                     ],
                                    _Button,
                                    Goal).

centered_message(List) :-
        centered_modal_dialog(65, 400, [     button(20, 340, 20, 50, ok.res),
                                              text(6, 55, 48, 275, wseq(List)),
                                              icon(17, 14, 32, 32, 0)
                                     ],
                                    _Button,
                                    Goal).

my_dialog(Name, Down, In, Depth, Width, Format_List, Button) :-
        my_dialog(Name, Down, In, Depth, Width, Format_List, Button, true).
my_dialog(Name, OldDown, OldIn, OldDepth, OldWidth, Format_List, Button, Goal) :-
        maybe_get_text(Name, Name_text),
        (recall(Name_text, (Down, In, Depth, Width))
        ;Down = OldDown, In = OldIn, Depth = OldDepth, Width = OldWidth
        ),
        !,
        get_dialog_resources(Format_List, New_Format_List),
        dialog(Name_text, Down, In, Depth, Width, New_Format_List, Button, my_call(Goal,
Name_text, NewDown, NewIn, NewDepth, NewWidth)),
        remember(Name_text, (NewDown, NewIn, NewDepth, NewWidth)).

my_mdialog(Down, In, Depth, Width, Format_List, Button) :-
        get_dialog_resources(Format_List, New_Format_List),
        mdialog(Down, In, Depth, Width, New_Format_List, Button).

my_call(_, _, Goal, Name, Down, In, Depth, Width) :-
        wsize(Name, Down, In, Depth, Width),
        Goal.

my_old(File, Vol) :-
        maybe_get_text(select_text_file.res, Select_text_file),
        old('TEXT', File, Vol, Select_text_file), !,
        detroit_font.
my_old(_, _) :-
        detroit_font,
        fail.

my_new(File, Vol, Window) :-
        maybe_get_text(save_as.res, Save_as),
        new(File, Vol, Save_as, Window), !,
        detroit_font.

my_old(_, _, _) :-
        detroit_font,
        fail.
```

```
proof_write(Window, Axiom, _Depth, []) :-
        write_terminator(Window, Axiom),
        find_next_problem(Window).
proof_write(Window, Tactic, Depth, Sequents) :-
        maybe_get_text(is_current_problem.res, Current_Problem),
        wsearch(Window, Current_Problem, 0, From, To),
        cursor(Window, From, To),
        nl_b(Window),
        depth_write(Window, Depth),
        write_b(Window, tactic_of.res),
        write_b(Window, Tactic),
        write_sequents(Window, Sequents, Depth).

write_sequents(Window, [First_Sequent | Rest], Depth) :-
        sequent_write(Window, First_Sequent, Depth),
        cursor_b(Window, A, _),
        write_b(Window, is_current_problem.res),
        cursor_b(Window, _, B),
        write_other_sequents(Window, Rest, Depth),
        cursor_b(Window, A, B).

write_other_sequents(_, [], _).
write_other_sequents(Window, [First | Rest], Depth) :-
        sequent_write(Window, First, Depth),
        write_other_sequents(Window, Rest, Depth).

sequent_write(Window, Sequent, Depth) :-
        nl_b(Window),
        depth_write(Window, Depth),
        write_b(Window, ' '),
        unparse_sequent(Sequent, Unparsed_Sequent),
        write_b(Window, Unparsed_Sequent).

depth_write(_Window, 0) :- !.
depth_write(Window, Depth) :-
        write_b(Window, '-'),
        NewDepth is Depth - 1,
        depth_write(Window, NewDepth).

write_problem_abandonned :-
        proof_window(Proof),
        cursor(Proof, 0, -1),
        write(Proof, 'Problem Abandoned').

find_next_problem(Window) :-
        cursor_b(Window, _, A),
        A1 is A + 1,
        wsearch(Window, '~M', A1, _, B),
        B1 is B - 1,
        cursor_b(Window, B1, B1),
        write_b(Window, is_current_problem.res),
        cursor_b(Window, End, End),
        cursor_b(Window, B1, End).
%find_next_problem(_).

write_terminator(Window, Name_of_axiom) :-
        write_b(Window, '■ '),
        write_b(Window, Name_of_axiom).
```

```
congratulate_user(Window) :-
        write_b(Window, well_done.res),
        bring_to_front(Window),
        minuses(Window).

minuses(Window) :-
  wsize(Window, _, _, D, W),
  'gdraw%i'(56, '-', Size),
  Times is (W-16) // Size,   % 16 is Scroll-bar width
  iterate(Times, write(Window, '-')).

iterate( 0, G) :- !.
iterate( N, G) :- N1 is N-1, G, iterate(N1, G).

write_b(Window, Text.res) :- !,
        maybe_get_text(Text.res, Real_text),
        write_b(Window, Real_text).
write_b(Window, Text) :-
        cursor(Window, From, To),
        wsltxt(Window, From, To, Oldtext),
        write(Window, Text),
        cursor(Window, _, Newto),
        asserta(written(Oldtext, Newto, From, To)).
write_b(Window, Text) :-
        retract(written(Oldtext, Newto, From, To)),
        cursor(Window, From, Newto),
        write(Window, Oldtext),
        cursor(Window, From, To), !,
        fail.

cursor_b(Window, From, To) :-
        cursor(Window, Oldfrom, Oldto),
        cursor(Window, From, To),
        asserta(cursored(Oldfrom, Oldto)).
cursor_b(Window, From, To) :-
        retract(cursored(Oldfrom, Oldto)),
        cursor(Window, Oldfrom, Oldto), !,
        fail.

nl_b(Window):-
        write_b(Window, '~M').
```

```
split_cont([], _, _, [], []) :- !.
split_cont(List, S1, S2, Selected_formula_Parsed, L2) :-
        current_logic(Logic),
        get_probs_initial(Logic, List, S1, S2, P1, P2),
        unparse_list(List, List_Unparsed),
        my_dialog2(select_formula_text.res, 40, 0, 240, 410,
                                        [button(207, 200, 23, 90, ok.res),
                                        button(207, 310, 23, 90, cancel.res),
                                        button(207, 10, 23, 170, split_context_dialog1.res),
                                        text(50, 200, 18, 100, split_context_dialog2.res),
                                        text(70, 200, 50, 200, P1),
                                        text(130, 200, 18, 117, split_context_dialog3.res),
                                        text(150, 200, 50, 200, P2),
                                        menu(50, 10, 150, 170, List_Unparsed,
                                        [], Selected_formula_Unparsed),
                                        text(10, 10, 40, 400, split_context_dialog4.res)],
                                              _, write_probs2(Logic, S1, S2, List_Unparsed,
Selected_formula_Parsed, L2)),
        parse_list(Selected_formula_Unparsed, Selected_formula_Parsed),
        split_list(List_Unparsed, Selected_formula_Unparsed, Ln2),
        parse_list(Ln2, L2), !.

write_probs2(Dialog, 3, Logic, Problem1, Problem2, All_items, Parsed_Selected_Items,
Parsed_Rest_Items) :- !,
        getditem(Dialog, 8, _, _, _, Selected_Items, _),
        parse_list(Selected_Items, Parsed_Selected_Items),
        parse_list(All_items, All_Parsed_Items),
        transform_problems(Logic, [Problem1], [NewProblem1]),
        unparse_sequent(NewProblem1, UnparsedProblem1),
        setditem(Dialog, 5, UnparsedProblem1), !,
        split_list(All_Parsed_Items, Parsed_Selected_Items, Parsed_Rest_Items),
        transform_problems(Logic, [Problem2], [NewProblem2]),
        unparse_sequent(NewProblem2, UnparsedProblem2),
        setditem(Dialog, 7, UnparsedProblem2), !,
        fail.

write_probs2(Dialog, 1, Logic, Problem1, Problem2, All_items, Parsed_Selected_Items,
Parsed_Rest_Items).

split_list(List, [], List).
split_list(List, [First | Rest], List2) :-
        remove(First, List, NewList),
        split_list(NewList, Rest, List2).
```

```
perform_cut(List, S1, S2, L1, L2, D) :-
        current_logic(Logic),
        get_probs_initial2(Logic, List, S1, S2, P1, P2, D),
        unparse_list(List, Nlist),
        my_dialog2(perform_cut.res, 40, 0, 310, 410,
                                        [button(277, 200, 23, 90, ok.res),
                                        button(277, 310, 23, 90, cancel.res),
                                        button(277, 10, 23, 170, split_context_dialog1.res),
                                        text(120, 200, 18, 100, split_context_dialog2.res),
                                        text(140, 200, 50, 200, P1),
                                        text(200, 200, 18, 117, split_context_dialog3.res),
                                        text(220, 200, 50, 200, P2),
                                        menu(120, 10, 150, 170, Nlist,
                                        [], Ln1),
                                        edit(80, 200, 18, 180, bytes("A"),
                                                        bytes(Cut_Formula)),
                                        text(80, 50, 18, 100, cut_formula.res),
                                        text(10, 10, 60, 400, perform_cut_header.res)],
                                                _, write_probs3(Logic, S1, S2, Nlist, L1, L2,
                                                                D, Cut_Formula)), !.


get_probs_initial(Logic, List, Prob1, Prob2, Initial_prob1, Initial_prob2) :-
        copy_term(Prob1, Copy_Prob1),
        varsin(Copy_Prob1, Z),
        Z = [[]],
        copy_term(Prob2, Copy_Prob2),
        varsin(Copy_Prob2, Z1),
        Z1 = [List],
        transform_problems(Logic, [Copy_Prob1], [New_Prob1]),
        transform_problems(Logic, [Copy_Prob2], [New_Prob2]),
        unparse_sequent(New_Prob1, Initial_prob1),
        unparse_sequent(New_Prob2, Initial_prob2).

my_dialog2(Name, Down, In, Depth, Width, Format_List, Button, Goal) :-
        maybe_get_text(Name, Name_text),
        get_dialog_resources(Format_List, New_Format_List),
        dialog(Name_text, Down, In, Depth, Width, New_Format_List, Button, Goal).
```

FILE AND WINDOW HANDLING

```
% Windows are created as program windows so that they have a goaway box
% They are given <USER1> (which always succeeds) as their compile mode to avoid poblems
during
% program development.

create_window(Name) :-
        wpcreate(Name, 1, '<USER1>', '<INTERPRET>', 45, 150, 300, 350),
        default_font_size(Size),
        wfont(Name, 'Konstanz', 0, Size),        % Set font-size to default
        wclchg(Name).  % Clear the edit flag

create_utility_window(Name) :-
        wpcreate(Name, 1, '<USER1>', '<INTERPRET>', 245, 150, 100, 350),
        default_font_size(Size),
        wfont(Name, 'Konstanz', 0, Size),        % Set font-size to default
        wclchg(Name).  % Clear the edit flag

'<USER1>'(_, _).

lock_proof_window :-
        proof_window(Proof),
        wlock(Proof, on).

unlock_proof_window :-
        proof_window(Proof),
        wlock(Proof, off).

clear_proof_window :-
        proof_window(Window),
        cursor(Window, 0, -1),
        clear(Window),
        write(Window, '~M~M~M'),
        cursor(Window, 0, 0).

refresh_front_window :-
        current_window(Window),
        whide(Window),
        wshow(Window),
        wfront(Window).

hide_front_window :-
        current_window(Window),
        whide(Window).

% Returns current window, just as long as it is really a window.
current_window(Window) :-
        wfront(Window),
        wtype(Window, prog).

bring_to_front(Window) :-
        is_win(Window, _),
        wfront(Window).
```

```
doesnt_already_exist(_, _, Window) :-             % Extra parameters added by dialog
handler.
        doesnt_already_exist(Window).
doesnt_already_exist(Window) :-
        not(is_win(Window, _)).
doesnt_already_exist(Window) :-
        that_window_already_exists_message(Window),
        fail.

no_conflict(_, _, Window, Window).     % Name has not been changed
no_conflict(_, _, _, NewWindow) :-
        not(is_win(NewWindow, _)).
no_conflict(_, _, _, Window) :-
        that_window_already_exists_message(Window),
        fail.

maybe_save_windows :-
        my_get_items(windows.menu, Items),
        Items = [_, _, _, _, _, _ | Windows],
        map(maybe_save, Windows).

maybe_save(Window) :-
        wchg(Window),
        ask_save_changes(Window),
        my_new(File, Vol, Window).
maybe_save(_).

not_locked(Window) :-
        wlock(Window, OnOff),
        OnOff = off.

proof_window_to_front :-
        proof_window(Proof),
        (bring_to_front(Proof), !
        ;create_window(Proof),
        my_extend_menu(windows.menu, [Proof]),
        actdeact(Proof, mark_menu_items)
        ).

read_problem(Assums ⊢ Goal, Bytes) :-
        wfront(Window),
        cursor(Window, From, To),
        To > From, !,
        reading_and_parsing_banner(read_and_parse(Window, From, To, Assums, Goal,
Bytes)).

read_problem(Problem, _) :-
        no_text_selected_message, !,
        fail.

read_and_parse(Window, From, To, Assums, Goal, Bytes) :-
        wsltxt(Window, From, To, Text),
        name(Text, Bytes),
        parse_problem(Bytes, Assums, Goal).
```

```
write_status :-
        status_window(Status),
        is_win(Status, _),
        cursor(Status, 0, -1),

        write_res(Status, 'Current Logic: '),
        current_logic(Logic),
        write(Status, Logic),

        check_theorem_prover(TP_Status),
        write(Status, '~MTheorem Prover '),

        (TP_Status = on -> write_res(Status, 'set to Search Depth '),

                                                search_depth(Depth),

                                        write(Status, Depth)
        ; otherwise      -> write(Status, 'off')
        ),

        last_problem(Prob),
        name(P, Prob),

        ( proof_editing(on)        -> write(Status, '~MCurrently Solving: ')
        ; otherwise                -> write(Status, '~MPrevious Problem: ')
        ),
        write_res(Status, P),

        ( theorem_proving(on) -> write(Status, '~MCurrently Theorem Proving')
        ; otherwise
        ).
write_status.


my_load(File, Vol) :-
        tload(File, Vol),
        wrename(File, 'temp%%%%'),  % Creates temporary display window
        create_window(File),
        move('temp%%%%', File),  %  Moves contents of display window into a program window
        wkill('temp%%%%').

move(Window, Window1) :-
        wlen(Window, Length),
        copy_bytes(Length, Window, Window1).

save_window :-
        current_window(Window),
        my_new(File, Vol, Window),
        tsave(File, Vol, [Window]),
        wclchg(Window).
```

DATABASE

```
set_default_font_size(Size) :- remember(default_font_size, Size).
default_font_size(Size) :- recall(default_font_size, Size).

set_default_window_name(Name) :- init_gensym(Name), assert(is_generator(Name)).
get_default_window_name(Name) :- is_generator(Generator), gensym(Generator, Name).

set_all_tactics :- remember(tactics, all), write_status.
set_applicable_tactics :- remember(tactics, applicable), write_status.
all_tactics_wanted :- recall(tactics, all).
applicable_tactics_wanted :- recall(tactics, applicable).

turn_theorem_prover(Status) :- remember(theorem_prover_status, Status), write_status.
check_theorem_prover(Status) :- recall(theorem_prover_status, Status).

set_search_depth(Depth) :- remember(search_depth, Depth), write_status.
search_depth(Depth) :- recall(search_depth, Depth).

set_current_logic(Logic) :- remember(current_logic, Logic), write_status.
current_logic(Logic) :- recall(current_logic, Logic).

set_last_sequent_in_dialog(Ass, Goal) :- remember(last_sequent_in_dialog, (Ass, Goal)).
last_sequent_in_dialog(Ass, Goal) :- recall(last_sequent_in_dialog, (Ass, Goal)).

set_axioms_as_tactics(on) :- remember(axioms_as_tactics, on), write_status.
set_axioms_as_tactics(off) :- remember(axioms_as_tactics, off), write_status.
axioms_as_tactics_set :- recall(axioms_as_tactics, on).

set_last_problem(Problem) :- remember(last_problem, Problem).
last_problem(Problem) :- recall(last_problem, Problem).

remember_theorem_proving_on :- remember(theorem_proving, on), write_status.
remember_theorem_proving_off :- remember(theorem_proving, off), write_status.
theorem_proving(A) :- recall(theorem_proving, A).

remember_proof_editing_on :- remember(proof_editing, on), write_status.
remember_proof_editing_off :- remember(proof_editing, off), write_status.
proof_editing(A) :- recall(proof_editing, A).

clear_database :-
        abolish(cursored, 2),
        abolish(written, 4).
```

OPERATORS AND INITIAL SETTING UP

```
:-
        op(120, xfx, '⊢'),                % ILL turnstyle
        op(120, fy, '⊢'),                 % CLL turnstyle

        op(300, xfx, '.'),                % Used to identify resources

        % LL connectives
        op(100, xfx, [→, ⊸, ∨, ⅋, ⊗]),
        op(99, xfx, [⊕, &]),
        op(100, fx, '¬'),
        op(90, fx, '!'),
        op(100, fx, '?'),
        op(98, xf, '⊥'),


        op(70, xf, '+'),
        op(70, xf, '-'),

        op(500, fx, '⟦'),
        op(501, xf, '⟧'),

        op(150, xfx, ':'),                % Used by theorem prover

        % Tactic and Axiom Stuff
        op(110, xfx, '/'),
        op(110, xfx, '^'),
        op(110, xfx, '::'),
        op(111, xfx, '-'),
        op(110, xfx, '∊'),
        op(111, xfx, 'equ'),
        op(110, xfx, 'plus'),
        op(131, xfx, 'given'),
        op(130, xfy, 'and'),

        op(140, xfx, 'is_axiomatic_if'),
        op(140, xf, 'is_axiomatic'),
        op(200, xfx, 'axiom_of'),
        op(150, xfx, ':'),        % Used by Axioms
        op(100, yf, 'is_empty'),

        op(140, xfx, 'is_a_syntactic_extension_of'),

        op(135, xfx, 'tactic_of'),
        op(145, xfx, 'converts'),
        op(135, xfx, 'where'),
        op(136, xfx, 'into'),
        op(144, xfx, 'if'),

        op(100, yf, 'is_cut_formula'), % TEMP
        op(100, yf, 'is_an_atom').
```

```
'<LOAD>'(_) :-
        maybe_setup_lpa_menu,
        set_up_database,
        create_status_window,
        create_initial_windows,
        install_menus,
        detroit_font.

maybe_setup_lpa_menu :-
        in_programming_environement,
        Menus = ['File', 'Edit', 'Fonts'],
        form_menu_list(Menus, Mlist),
        install_menu('LPA', Mlist).
maybe_setup_lpa_menu.

in_programming_environement :- is_menu('Search').

form_menu_list([], []) :- !.
form_menu_list([Menu | Rest1], [Menu(Items) | Rest2] ) :-
        get_items(Menu, Items), !,
        form_menu_list(Rest1, Rest2).

% my_assert is used because menu items may end with / followed by a capital letter
associating a control key to that menu item.
% These two characters if present, need to be removed.

set_up_database :-
        set_default_font_size(9),
        remember_proof_editing_off,
        remember_theorem_proving_off,
        set_applicable_tactics,
        turn_theorem_prover(on),
        set_search_depth(0),
        maybe_get_text(window.res, Window),
        set_default_window_name(Window),

        set_last_sequent_in_dialog("A&B", "A").

create_status_window :-
        succeed(wkill('Status')),
        status_window(Status),
        create_utility_window(Status),
        actdeact(Status, mark_menu_items).

create_initial_windows :-
        succeed(wkill('Proof')),
        succeed(wkill('Jotter')),
        proof_window(Proof),
        jotter_window(Jotter),
        create_window(Jotter),
        create_window(Proof),
        actdeact(Jotter, mark_menu_items),
        actdeact(Proof, mark_menu_items).
```

MENU HANDLING

```
install_edit_menu :-
        get_menu_header(edit.menu, Edit),
        get_menu_item(undo, Undo),
        get_menu_item(cut, Cut),
        get_menu_item(copy, Copy),
        get_menu_item(paste, Paste),
        get_menu_item(clear, Clear),
        get_menu_item(balance, Balance),
        get_menu_item(select_all, Select_all),
        get_menu_item(keystrokes, Keystrokes),
        install_menu(Edit, [Undo, '(-', Cut, Copy, Paste, Clear, '(-', Balance, Select_all, '(-',
        Keystrokes(['&', '∨', '→', '~', '∧', '⊥', 'T', '⊦', '⊸', '⊕', '⊗', '⅋', '!', '⊥'])
        ]),
        menu_assert((Edit(Balance) :- edit(balance))),
        menu_assert((Edit(Undo) :- edit(undo))),
        menu_assert((Edit(Cut) :- edit(cut))),
        menu_assert((Edit(Copy) :- edit(copy))),
        menu_assert((Edit(Select_all) :- edit(select_all))),
        menu_assert((Edit(Clear) :- edit(clear))),
        menu_assert((Edit(Paste) :- edit(paste))),
        assert((Keystrokes(A) :- keystrokes(A))).

edit(balance) :-
        wfront(Window),
        balance(Window).

edit(Command) :-
        'front%i'(Window),
        'wtype%i'(Window, Type),
        edit_command(Command, Window, Type).

edit_command(Command, Window, Type) :-
        on(Type, [10, 12, 13, 14]),
        !,
        'gttl%i'(Window, WindowName),
        ((Command = 'undo';Command = 'clear';Command = 'paste';Command = 'cut') ->
        not_locked(WindowName)
        ; true),
        Command(WindowName).

edit_command(undo, Window, 2) :-  'undo%i'(Window).
edit_command(cut, Window, 2) :-  not_locked(Window), 'cut%i'(Window).
edit_command(copy, Window, 2) :- 'copy%i'(Window).
edit_command(paste, Window, 2) :-  'paste%i'(Window).
edit_command(clear, Window, 2) :-  'clear%i'(Window).

select_all(Window) :-
        cursor(Window, 0, -1).

keystrokes(Character) :-
        copy_to_window(Character).
copy_to_window(Character) :-
        wfront(Window),
        not_locked(Window),
        wtype(Window, prog),
        write(Window, Character).
copy_to_window(_) :-
        wfront(Window),
        not_locked(Window),
        wtype(Window, dial),
        that_doesnt_work_on_dialogs_message,
        !,
        fail.
install_file_menu :-
        (in_programming_environement -> File = 'File '  ;  get_menu_header(file.menu, File)),
% So that multifinder can shut down application
        get_menu_item(open.m_item, Open),
        get_menu_item(save.m_item, Save),
        get_menu_item(pagesetup.m_item, Pagesetup),
        get_menu_item(print.m_item, Print),
        get_menu_item(reset.m_item, Reset),
        get_menu_item(quit.m_item, Quit),
        install_menu(File, [Open, Save, '(-', Pagesetup, Print, '(-', Reset, Quit]),
        menu_assert((File(Open) :- file(open))),
        menu_assert((File(Save) :- file(save))),
        menu_assert((File(Pagesetup) :- call(pgsetup))),
        menu_assert((File(Print) :- file(print))),
        menu_assert((File(Reset) :- file(reset))),
        menu_assert((File(Quit) :- file(quit))).

file(open) :-
        my_old(File, Vol),
        (doesnt_already_exist(File);    % Ok
        file(open)          % Try again
        ),
        cursor(crane),
        quote(File, File_txt),
        loading_file_banner(File, my_load(File, Vol)),
        my_extend_menu(windows.menu, [File]),
        actdeact(File, mark_menu_items).

file(save) :-
        save_window.

file(print) :-
        current_window(Window),
        printwins([Window]).

file(reset) :-
        reenable_menus,
        clear_database,
        unlock_proof_window.

file(quit) :-
        ask_really_quit,
        maybe_save_windows,
        reset_system_font,
        halt.
```

```
install_font_menu :-
        get_menu_header(fonts.menu, Fonts),
        get_menu_header(default_size.menu, Default_font_size),
        install_menu(Fonts, [Default_font_size(['9','12'])]),
        assert((Default_font_size(Size) :- default_size(Size))),
        add_fonts(Fonts),
        get_menu_item(normal.m_item, Normal),
        get_menu_item(bold.m_item, Bold),
        get_menu_item(italic.m_item, Italic),
        get_menu_item(underline.m_item, Underline),
        get_menu_item(outline.m_item, Outline),
        get_menu_item(shadow.m_item, Shadow),
        extend_menu(Fonts, ['(-', Normal, Bold, Italic, Underline, Outline, Shadow, '(-', '9',
'10', '12', '14', '18']),
        style_item(Fonts, Normal, normal),
        style_item(Fonts, Bold, bold),
        style_item(Fonts, Italic, italic),
        style_item(Fonts, Underline, underline),
        style_item(Fonts, Outline, outline),
        style_item(Fonts, Shadow, shadow),
        assert((Fonts(Item) :- fonts(Item))),
        mark_item(Default_font_size, '9'),
        (is_font('Konstanz')
        ;centered_yes_no([konstanz_not_installed.res])
        ),
        (is_font('Detroit')
        ;centered_yes_no([detroit_not_installed.res])
        ).

fonts(Font) :-
        is_font(Font),
        current_window(Window),
        wfont(Window, Font, _Face, _Size),
        unmark_fonts,
        my_mark_item(fonts.menu, Font).

fonts(Size_text) :-
        pname(Size, Size_text),
        integer(Size),
        current_window(Window),
        wfont(Window, _Font, _Face, Size),
        unmark_sizes,
        my_mark_item(fonts.menu, Size_text).

fonts(Face_text) :-
        current_window(Window),
        wfont(Window, _Font, Old_Face_number, _Size),
        get_menu_item(Face.m_item, Face_text),
        get_face_number(Face, Bit, Number),
        (Number = 0 -> Face_number = 0
        ;btest(Old_Face_number, Bit) -> Face_number is Old_Face_number - Number
        ;otherwise -> Face_number is Old_Face_number + Number),
        wfont(Window, _Font, Face_number, _Size),
        mark_styles(Face_number).
```

```
unmark_fonts :-
        my_get_items(fonts.menu, Items),
        map(unmark_font, Items).

unmark_font(Item) :-
        is_font(Item),
        my_unmark_item(fonts.menu, Item).
unmark_font(_).

unmark_sizes :-
        map(my_unmark_item(fonts.menu), ['9', '10', '12', '14', '18']).

get_face_number(normal, -1, 0).
get_face_number(bold, 0, 1).
get_face_number(italic, 1, 2).
get_face_number(underline, 2, 4).
get_face_number(outline, 3, 8).
get_face_number(shadow, 4, 16).

default_size('9') :-
        set_default_font_size(9),
        toggle(default_size.menu, '9', ['12']).

default_size('12') :-
        set_default_font_size(12),
        toggle(default_size.menu, '12', ['9']).

mark_menu_items(activate, Window) :-
        wfont(Window, Font, Style, Size),
        my_mark_item(fonts.menu, Font),
        pname(Size, Size_text),
        my_mark_item(fonts.menu, Size_text),
        mark_styles(Style),
        mark_window(Window).

mark_menu_items(deactivate, Window) :-
        my_get_items(fonts.menu, Items),
        map(my_unmark_item(fonts.menu), Items),
        my_unmark_item(windows.menu, Window).

mark_window(Window) :-
        my_mark_item(windows.menu, Window).

mark_styles(0) :-
        get_menu_item(normal.m_item, Item_text),
        my_mark_item(fonts.menu, Item_text),
        map(my_unmark_item(fonts.menu),

                              [bold.m_item, italic.m_item, underline.m_item, outline.m_item,
shadow.m_item]).
mark_styles(Style) :-
        forall(get_face_number(Item, Bit, _Number), test_and_mark(Style, Item, Bit)).

test_and_mark(Style, Item, Bit) :-
        btest(Style, Bit),
        my_mark_item(fonts.menu, Item.m_item).
test_and_mark(Style, Item, _) :-
        my_unmark_item(fonts.menu, Item.m_item).
```

```
install_logic_menu :-
        get_menu_header(logic.menu, Logic),
        get_all_logics(Logics),
        install_menu(Logic, Logics),
        assert((Logic(A) :- logic(A))),
        Logics = [First | _Rest],
        mark_item(Logic, First),
        set_current_logic(First).

logic(Logic) :-
        current_logic(Old_Logic),
        my_unmark_item(logic.menu, Old_Logic),
        my_mark_item(logic.menu, Logic),
        set_current_logic(Logic).
install_options_menu :-
        get_menu_header(options.menu, Options),
        get_menu_item(all_tactics.m_item, All_tactics),
        get_menu_item(applicable_tactics.m_item, Applicable_tactics),
        get_menu_item(theorem_prover_on.m_item, Theorem_prover_on),
        get_menu_item(theorem_prover_off.m_item, Theorem_prover_off),
        get_menu_item(axioms_as_tactics_on.m_item, Axioms_as_tactics_on),
        get_menu_item(axioms_as_tactics_off.m_item, Axioms_as_tactics_off),
        get_menu_header(depth_of_search.menu, Depth_of_search),
        install_menu(Options, [All_tactics, Applicable_tactics, '(-', Axioms_as_tactics_on,
Axioms_as_tactics_off, '(-', Theorem_prover_on, Theorem_prover_off,
Depth_of_search(['0', '1', '2', '3', '4', '5', '6', '7'])]),
        menu_assert((Options(All_tactics) :- options(all_tactics))),      % my_window used so
as not to conflict with inbuilt window predicate
        menu_assert((Options(Applicable_tactics) :- options(applicable_tactics))),
        menu_assert((Options(Axioms_as_tactics_on) :- options(axioms_as_tactics_on))),
        menu_assert((Options(Axioms_as_tactics_off) :- options(axioms_as_tactics_off))),
        menu_assert((Options(Theorem_prover_on) :- options(theorem_prover_on))),
        menu_assert((Options(Theorem_prover_off) :- options(theorem_prover_off))),
        assert((Depth_of_search(A) :- depth_of_search(A))),
        mark_item(Options, Theorem_prover_on),
        mark_item(Options, Applicable_tactics),
        mark_item(Depth_of_search, '0'),
        options(axioms_as_tactics_off).

options(all_tactics) :-
        set_all_tactics,
        toggle(options.menu, all_tactics.m_item, [applicable_tactics.m_item]).

options(applicable_tactics) :-
        set_applicable_tactics,
        toggle(options.menu, applicable_tactics.m_item, [all_tactics.m_item]).

options(axioms_as_tactics_on) :-
        toggle(options.menu, axioms_as_tactics_on.m_item,
[axioms_as_tactics_off.m_item]),
        set_axioms_as_tactics(on).

options(axioms_as_tactics_off) :-
        toggle(options.menu, axioms_as_tactics_off.m_item,
[axioms_as_tactics_on.m_item]),
        set_axioms_as_tactics(off).
```

```
options(theorem_prover_on) :-
        turn_theorem_prover(on),
        toggle(options.menu, theorem_prover_on.m_item, [theorem_prover_off.m_item]),
        get_menu_header(depth_of_search.menu, Depth_of_Search),
        my_enable_item(options.menu, Depth_of_Search).

options(theorem_prover_off) :-
        turn_theorem_prover(off),
        toggle(options.menu, theorem_prover_off.m_item, [theorem_prover_on.m_item]),
        get_menu_header(depth_of_search.menu, Depth_of_Search),
        my_disable_item(options.menu, Depth_of_Search).


depth_of_search(Depth) :-
        map(my_unmark_item(depth_of_search.menu), ['0', '1', '2', '3', '4', '5', '6', '7']),
        my_mark_item(depth_of_search.menu, Depth),
        pname(Depth_integer, Depth),
        set_search_depth(Depth_integer).
install_problem_menu :-
        get_menu_header(problem.menu, Problem),
        get_menu_item(dialog.m_item, Dialog),
        get_menu_item(front_window.m_item, Front_window),
        get_menu_item(same_again.m_item, Same_again),
        install_menu(Problem, [Dialog, Front_window, Same_again]),
        menu_assert((Problem(Dialog) :- problem(dialog))),
        menu_assert((Problem(Same_again) :- problem(same_again))),
        menu_assert((Problem(Front_window) :- problem(front_window))).

problem(dialog) :-
        disable_menus1,
        current_logic(Logic),
        enter_problem(Logic, Problem, Bytes),
        set_last_problem(Bytes),
        start_problem(Problem).
problem(dialog) :-
        reenable_menus.

problem(same_again) :-
        last_problem(Bytes),
        parse_problem_same_again(Bytes, Assumsions, Goal),
        start_problem(Assumsions ⊢ Goal).

problem(front_window) :-
        read_problem(Problem, Bytes),
        set_last_problem(Bytes),
        start_problem(Problem).
```

```
install_windows_menu :-
        get_menu_header(windows.menu, Windows),
        get_menu_item(create.m_item, Create),
        get_menu_item(refresh.m_item, Refresh),
        get_menu_item(rename.m_item, Rename),
        get_menu_item(hide.m_item, Hide),
        get_menu_item(kill.m_item, Kill),
        proof_window(Proof),
        jotter_window(Jotter),
        status_window(Status),
        install_menu(Windows, [Create, Refresh, Rename, Hide, Kill, '(-']),
        menu_assert((Windows(Create) :- my_windows(create))),       % my_window used so
                                            as not to conflict with inbuilt window predicate
        menu_assert((Windows(Refresh) :- my_windows(refresh))),
        menu_assert((Windows(Rename) :- my_windows(rename))),
        menu_assert((Windows(Hide) :- my_windows(hide))),
        menu_assert((Windows(Kill) :- my_windows(kill))),
        assert((Windows(Window) :- my_windows(Window))),
        proof_window(Proof),
        jotter_window(Jotter),
        my_extend_menu(windows.menu, [Jotter, Proof, Status]).

my_windows(create) :-
        get_default_window_name(Window),
        get_window_name(Window, New_Window_Name,


        doesnt_already_exist(New_Window_Name)),
        create_window(New_Window_Name),
        my_extend_menu(windows.menu, [New_Window_Name]),
        actdeact(New_Window_Name, mark_menu_items).

my_windows(refresh) :-
        refresh_front_window.

my_windows(rename) :-
        current_window(Window),
        enter_new_window_name(Window, New_Name,


        no_conflict(Window, New_Name)),
        wrename(Window, New_Name),
        rename_menu_item(windows.menu, Window, New_Name).

my_windows(hide) :-
        hide_front_window.

my_windows(kill) :-
        current_window(Window),
        ask_really_kill_window(Window),
        wkill(Window),
        remove_window_from_menu(Window).

my_windows(Window) :-
        bring_to_front(Window).
```

```
% Take menu appart and rebuild it.
remove_window_from_menu(Window) :-
        my_get_items(windows.menu, Items),
        Items = [_, _, _, _, _, _ | Window_Items],
        remove(Window, Window_Items, Rest_Windows),
        get_menu_header(windows.menu, Windows),
        get_menu_item(create.m_item, Create),
        get_menu_item(refresh.m_item, Refresh),
        get_menu_item(rename.m_item, Rename),
        get_menu_item(hide.m_item, Hide),
        get_menu_item(kill.m_item, Kill),
        proof_window(Proof),
        jotter_window(Jotter),
        clear_menu(Windows),
        extend_menu(Windows, [Create, Refresh, Rename, Hide, Kill, '(-']),
        my_extend_menu(windows.menu, Rest_Windows).

disable_menus :-
        my_disable_menu(problem.menu),
        my_disable_menu(logic.menu).

disable_menus1 :-
        my_disable_menu(problem.menu).

reenable_menus :-
        my_enable_menu(problem.menu),
        my_enable_menu(logic.menu).

my_disable_menu(Menu.menu) :-
        get_menu_header(Menu.menu, Menu_header),
        disable_menu(Menu_header).

my_enable_menu(Menu.menu) :-
        get_menu_header(Menu.menu, Menu_header),
        enable_menu(Menu_header).

my_extend_menu(Menu, Item_List) :-
        get_menu_header(Menu, Menu_Header),
        map(maybe_get_menu_item, Item_List, Items_text),
        extend_menu(Menu_Header, Items_text).

rename_menu_item(Menu, Item, NewItem) :-
        get_menu_header(Menu, Menu_Header),
        rename_item(Menu_Header, Item, NewItem).

my_get_items(Menu, Items) :-
        get_menu_header(Menu, Menu_Header),
        get_items(Menu_Header, Menu_Items),
        Menu_Items = Items.     % get_items behaves strangely

toggle(Menu, Item, ItemList) :-
        my_mark_item(Menu, Item),
        map(my_unmark_item(Menu), ItemList).
```

```
my_mark_item(Menu, Item) :-
        get_menu_header(Menu, Menu_Header),
        maybe_get_menu_item(Item, Items_text),
        mark_item(Menu_Header, Items_text).

my_unmark_item(Menu, Item) :-
        get_menu_header(Menu, Menu_Header),
        maybe_get_menu_item(Item, Items_text),
        unmark_item(Menu_Header, Items_text).

my_disable_item(Menu, Item) :-
        get_menu_header(Menu, Menu_Header),
        maybe_get_menu_item(Item, Items_text),
        disable_item(Menu_Header, Items_text).

my_enable_item(Menu, Item) :-
        get_menu_header(Menu, Menu_Header),
        maybe_get_menu_item(Item, Items_text),
        enable_item(Menu_Header, Items_text).

install_menus :-
        install_file_menu,
        install_edit_menu,
        install_logic_menu,
        install_problem_menu,
        install_windows_menu,
        install_options_menu,
        install_font_menu.

menu_assert((Operator1(Item1) :- Operator2(Item2))) :-
        maybe_remove_control(Item1, NewItem1),
        assert((Operator1(NewItem1) :- !, Operator2(Item2))).

maybe_remove_control(Atom, NewAtom) :-
        name(Atom, String),
        maybe_remove_last_two_chars(String, NewString),
        name(NewAtom, NewString).

maybe_remove_last_two_chars([47, _], []):- !.          % 47 is ascii for /
maybe_remove_last_two_chars([Char | Rest], [Char | NewRest]) :-
        maybe_remove_last_two_chars(Rest, NewRest).
maybe_remove_last_two_chars([Char], [Char]).
```

RESOURCE HANDLING

```
get_dialog_resources(Format_List, New_Format_List) :-
        map(get_dialog_resource, Format_List, New_Format_List).

get_dialog_resource(text(Down_pos, In_pos, Depth, Width, Resource_name.res),
text(Down_pos, In_pos, Depth, Width, Text)) :- !,
        maybe_get_text(Resource_name.res, Text).

get_dialog_resource(text(Down_pos, In_pos, Depth, Width, wseq(List)), text(Down_pos,
In_pos, Depth, Width, wseq(NewList))) :- !,
        map(maybe_get_text, List, NewList).

get_dialog_resource(button(Down_pos, In_pos, Depth, Width, Resource_name.res),
button(Down_pos, In_pos, Depth, Width, Label)) :- !,
        maybe_get_text(Resource_name.res, Label).

get_dialog_resource(menu(Down_pos, In_pos, Depth, Width, Items, Presel, Sels),
menu(Down_pos, In_pos, Depth, Width, Items, Presel, Sels, 'Konstanz', Size, 0)) :- !,
        default_font_size(Size).

get_dialog_resource(Item, Item).

detroit_font :-
        succeed(call_pascal(149, 'MINE', 128)).

reset_system_font :-
        call_pascal('MINE', 128).

get_menu_header(file.menu, 'File ') :- in_programming_environement, !.

% These actualy appear in a resource file, but are included as code for clarity

get_menu_header(file.menu, 'File') :- !.
get_menu_header(edit.menu, 'Edit ') :- !.
get_menu_header(windows.menu, 'Windows ') :- !.
get_menu_header(logic.menu, 'Logic') :- !.
get_menu_header(problem.menu, 'Problem') :- !.
get_menu_header(options.menu, 'Options') :- !.
get_menu_header(fonts.menu, 'Fonts ') :- !.
get_menu_header(default_size.menu, 'Default Font Size') :- !.
get_menu_header(depth_of_search.menu, 'Depth of Search') :- !.
get_menu_header(problem.menu, 'Problem') :- !.

get_menu_header(Menu.menu, Menu_header) :-
        res_atom(Menu, Menu_header).

get_menu_item(create.m_item, 'Create...') :- !.
get_menu_item(refresh.m_item, 'Refresh') :- !.
get_menu_item(rename.m_item, 'Rename...') :- !.
get_menu_item(hide.m_item, 'Hide/H') :- !.
get_menu_item(kill.m_item, 'Kill...') :- !.
get_menu_item(undo, 'Undo/Z') :- !.
get_menu_item(cut, 'Cut/X') :- !.
get_menu_item(copy, 'Copy/C') :- !.
get_menu_item(paste, 'Paste/V') :- !.
get_menu_item(clear, 'Clear') :- !.
get_menu_item(balance, 'Balance/B') :- !.
get_menu_item(select_all, 'Select all/A') :- !.
get_menu_item(keystrokes, 'Keystrokes') :- !.

get_menu_item(normal.m_item, 'Normal') :- !.
get_menu_item(bold.m_item, 'Bold') :- !.
get_menu_item(italic.m_item, 'Italic') :- !.
get_menu_item(underline.m_item, 'Underline') :- !.
get_menu_item(outline.m_item, 'Outline') :- !.
get_menu_item(shadow.m_item, 'Shadow') :- !.

get_menu_item(open.m_item, 'Open text file.../O') :- !.
get_menu_item(save.m_item, 'Save to text file.../S') :- !.
get_menu_item(pagesetup.m_item, 'Page setup...') :- !.
get_menu_item(print.m_item, 'Print.../P') :- !.
get_menu_item(reset.m_item, 'Reset') :- !.
get_menu_item(quit.m_item, 'Quit/Q') :- !.

get_menu_item(all_tactics.m_item, 'All Tactics') :- !.
get_menu_item(applicable_tactics.m_item, 'Applicable Tactics') :- !.
get_menu_item(axioms_as_tactics_on.m_item, 'Axioms as tactics on') :- !.
get_menu_item(axioms_as_tactics_off.m_item, 'Axioms as tactics off') :- !.
get_menu_item(theorem_prover_on.m_item, 'Theorem Prover On') :- !.
get_menu_item(theorem_prover_off.m_item, 'Theorem Prover Off') :- !.

get_menu_item(same_again.m_item, 'Same Again/3') :- !.
get_menu_item(front_window.m_item, 'Front Window/2') :- !.
get_menu_item(dialog.m_item, 'Dialog.../1') :- !.


% These actualy appear in a resource file, but are included as code for clarity

maybe_get_menu_item(Text, Text).
maybe_get_text(enter_new_window_name.res, 'Enter new window name:') :- !.
maybe_get_text(status.res, 'Status') :- !.
maybe_get_text(window.res, 'Window') :- !.
maybe_get_text(select_text_file.res, 'Select a text file:') :- !.
maybe_get_text(ok.res, 'OK') :- !.
maybe_get_text(select_formula_text.res, 'Split Context') :- !.
maybe_get_text(select_one_dialogue.res, 'Formulas') :- !.
maybe_get_text(select_a_formula.res, 'Select a formula;') :- !.
maybe_get_text(cancel.res, 'Cancel') :- !.
maybe_get_text(continue.res, 'Continue') :- !.
maybe_get_text(split_context_dialog1.res, 'Show Problems') :- !.
maybe_get_text(split_context_dialog2.res, 'First Problem:') :- !.
maybe_get_text(split_context_dialog3.res, 'Second Problem:') :- !.
maybe_get_text(split_context_dialog4.res, 'Select part of context to be included in first
problem. Press "Show Problems" to see the resulting problems.') :- !.
maybe_get_text(perform_cut_header.res, 'Select part of context to be included in first
problem and type in the cut formula. Press "Show Problems" to see the resulting problems.') :-
!.
maybe_get_text(perform_cut.res, 'Cut') :- !.
maybe_get_text(that_doesnt_work_on_dialogs.res, 'Sorry, that doesn~'t work on dialogs.') :-
!.
```

```
maybe_get_text(change.res, 'Change') :- !.
maybe_get_text(no_text_selected.res, 'That only works if you have some text selected in the
front window!') :- !.
maybe_get_text(the_tactic_of.res, 'The tactic for ') :- !.
maybe_get_text(is_not_applicable.res, ' is not applicable.') :- !.
maybe_get_text(do_you_really_want_to_quit.res, 'Do you really want to quit?') :- !.
maybe_get_text(loading_the_file.res, 'Loading the file') :- !.
maybe_get_text(save_as.res, 'Save as:') :- !.
maybe_get_text(save_changes_to.res, 'Save changes to') :- !.
maybe_get_text(to.res, 'to:') :- !.
maybe_get_text(tactic_of.res, ' tactic for ') :- !.
maybe_get_text(yes.res, 'Yes') :- !.
maybe_get_text(select_a_tactic.res, 'Select a tactic') :- !.
maybe_get_text(or_operation.res, 'or operation;') :- !.
maybe_get_text(no.res, 'No') :- !.
maybe_get_text(tactic_choice.res, 'Tactic choice') :- !.
maybe_get_text(the_window.res, 'The window') :- !.
maybe_get_text(according_to_the_validity_checker.res, 'According to the validity checker')
:- !.
maybe_get_text(already_exists.res, 'already exists!') :- !.
maybe_get_text(provable_in.res, 'provable in ') :- !.
maybe_get_text(level.res, ' (level ') :- !.
maybe_get_text(question_mark.res, '?') :- !.
maybe_get_text(right_bracket.res, ').') :- !.
maybe_get_text(really_kill_window.res, 'Really kill the window') :- !.
maybe_get_text(proof.res, 'Proof') :- !.
maybe_get_text(stop.res, 'Stop') :- !.
maybe_get_text(is_current_problem.res, ' is current problem.') :- !.
maybe_get_text(multiplicative_and_additivelinear_logic.res, 'Multiplicative and Additive
Linear Logic') :- !.
maybe_get_text(intuitionistic_multiplicative_and_additivelinear_logic.res, 'Intuitionistic
Multiplicative and Additive Linear Logic') :- !.
maybe_get_text(intuitionistic_linear_logic.res, 'Intuitionistic Linear Logic') :- !.
maybe_get_text(classical_linear_logic.res, 'Classical Linear Logic') :- !.
maybe_get_text(il_sequent.res, 'IL sequent:') :- !.
maybe_get_text(translated_to_cll_sequent.res, 'Translated to CLL sequent: ') :- !.
maybe_get_text(translated_to_ill_sequent.res, 'Translated to ILL sequent: ') :- !.
maybe_get_text(translated_to_imall_sequent.res, 'Translated to IMALL sequent: ') :- !.
maybe_get_text(intuitionistic_logic_embedded_in_cll.res, 'Intuitionistic Logic embedded in
CLL') :- !.
maybe_get_text(intuitionistic_logic_embedded_in_ill.res, 'Intuitionistic Logic embedded in ILL')
:- !.
maybe_get_text(intuitionistic_logic_embedded_in_imall.res, 'Intuitionistic Logic embedded in
IMALL') :- !.
maybe_get_text(backtrack.res, 'Backtrack') :- !.
maybe_get_text(well_done.res, '~MWell Done~M') :- !.
maybe_get_text(jotter.res, 'Jotter') :- !.
maybe_get_text(may_not_be.res, 'may not be ') :- !.
maybe_get_text(cut_formula.res, 'Cut Formula:') :- !.
maybe_get_text(is_not.res, 'is not ') :- !.
maybe_get_text(detroit_not_installed.res, 'The font "Detroit" is not installed! This program
will be unintelligable without it. Do you wish to carry on?') :- !.
maybe_get_text(konstanz_not_installed.res, 'The font "Konstanz" is not installed! This
program will be unintelligable without it. Do you wish to carry on?') :- !.

/* maybe_get_text(Resource_Name.res, Text) :- !,
        res_atom(Resource_Name, Text). */
maybe_get_text(Text, Text).
```

```
is_backtrack(Backtrack) :-
        maybe_get_text(backtrack.res, Backtrack_text),
        Backtrack_text = Backtrack.
is_stop(Stop) :-
        maybe_get_text(stop.res, Stop).

proof_window(Proof) :- maybe_get_text(proof.res, Proof).
jotter_window(Jotter) :- maybe_get_text(jotter.res, Jotter).
status_window(Status) :- maybe_get_text(status.res, Status).
```