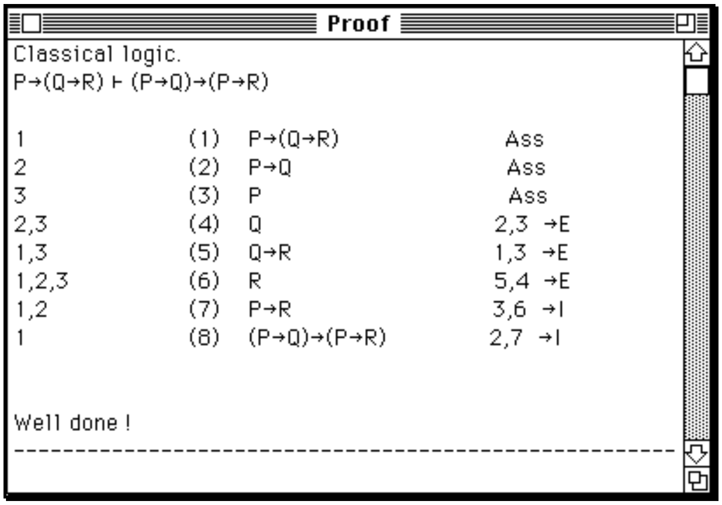
MacLogic

A Proof Assistant for First-Order Logic on the Macintosh

MacLogic is a proof assistant for first-order classical, intuitionistic and minimal logic, with or without equality, and either non-modal or modal (S4 or S5). It runs on any Apple Macintosh™ with at least 2 Mbyte RAM (an old version for 1Mby is available on request.) It works in two modes: as a proof checker and as a proof constructor. The proof checker verifies that a proof is indeed correct by checking it line by line in a “bottom-up” fashion. Proofs are laid out in the style of E. J. Lemmon’s *Beginning Logic*. However, the rules are not Lemmon’s, but much closer to Gentzen’s original natural deduction systems **NK**, **NJ** and **NM**. The proof constructor uses a “top-down” approach, based essentially on Gentzen’s sequent calculi such as **LJ**: once a proof has been constructed, it is transformed mechanically into one laid out in Lemmon’s style:



```
Classical logic.
P→(Q→R) ⊢ (P→Q)→(P→R)

1              (1)   P→(Q→R)            Ass
2              (2)   P→Q                Ass
3              (3)   P                  Ass
2,3            (4)   Q                 2,3  →E
1,3            (5)   Q→R               1,3  →E
1,2,3          (6)   R                 5,4  →E
1,2            (7)   P→R               3,6  →I
1              (8)   (P→Q)→(P→R)       2,7  →I


Well done !
-----------------------------------------------------
```

Problems may be entered by the user, read from a library file or read from a window on the screen. Complete proofs can be saved to disc and can be printed out either from within MacLogic itself, or by using any word-processor. Complete proofs can also be saved to a window: previous proofs from the current session can thus be inspected. Solved propositional problems may be kept as theorems for later use, during either the same or a later run of MacLogic.

Windows can be created, edited, saved, printed and killed as usual. They can be used for editing problems, constructing libraries of problems, or just making notes. So, MacLogic can be used as a text editor.

Theorem provers for various logics are included, to warn about attempts at unsolvable problems. They can be switched off if obtrusive.

An extensive on-line Help system reduces the need for availability of a manual beside the computer.

A special font, Konstanz, including a wide range of logical constants in addition to the standard Geneva character set, is included, both in bit-map form and in TrueType and PostScript type 1 format.

Contents

1. **Introduction**

2. **Getting Started**

3. **Advanced work**

4. **Theoretical background and bibliography**

5. **Reference guide**

1. Introduction

MacLogic is a tool intended to be used by the student learning to construct proofs in elementary logic. It is not a traditional piece of computer-based learning software, assigning marks for good work and so on: it is more like a word-processor than like a typing tutor. It has a modest amount of intelligence with which to warn you about incorrect or unpromising approaches to problems . One of the fascinations of logic, however, is that there are unsolvable problems: thus, MacLogic is not always able to show you what to do next!

By using MacLogic carefully and sensibly you will be able to learn a good deal about a variety of first-order logics. This manual will show you how to use MacLogic, and provides reference information to which you may refer when the on-line Help system incorporated in MacLogic is inadequate. In many cases, however, this on-line Help will give you as much information as you need, and thus you should soon be able to put this manual back on your shelf.

MacLogic is not intended to be, or to replace, an introductory text-book on logic, explaining the motivation, the notation, etc. Unfortunately, few text books are geared to the kind of computer-based approach that we espouse. One possible text on logic which can be used alongside MacLogic is that by Read & Wright: (see the bibliography in Section 4.5).

1.1 Elementary logic — its scope and purpose

By *elementary logic*, we understand certain aspects of logic which are regarded as fundamental. We avoid the use of compound terms, such as $x + y$, and stick to the part of logic concerned with purely logical rules rather than with rules deriving from mathematical practice, or from some other specific domain. This restriction allows us to use a fairly simple syntax, and to make certain parts of the MacLogic program itself fairly efficient.

We stick to zero-order and first-order logic, traditionally known as *propositional calculus* and *predicate calculus* respectively. (We prefer the non-traditional names, to suggest to the discerning reader that logicians are also interested in higher-order logics.)

MacLogic is a proof assistant, intended as a vehicle for learning about proofs. In our view, proof theory is fundamental to logic, and the “semantic” aspect of logic, concerning Boolean algebras, valuations, models, etc, is secondary. Proof theory owes a great deal to Gentzen’s pioneering work in the 1930s, especially as kept alive by the intuitionists and type theorists such as Girard, Heyting and Martin-Löf. Some so-called ‘semantic’ techniques are just notational variants of the syntactic notions we shall encounter.

1.2 Proof construction *vs* presentation

Many textbooks of logic implicitly aim to teach the art of presenting proofs in a certain format, the *natural deduction* style. That is to say, complete proofs are presented, beginning of course with simple proofs and moving on to more complex proofs. After a short while, the complexity of these proofs becomes overwhelming, and it is hard to see how they were obtained in the first place. Sometimes hints are given, of course. More often than not, no complete proofs are given after a certain stage, just instructions on how one could produce a complete proof if one had to. Some texts even present an even more forbidding format, the *axiomatic* style, often ascribed to Hilbert: presentation of complete proofs then ceases even earlier in the text. This style is convenient if one is going to do meta-logic, but is not one in which anyone ever constructs (non-trivial) proofs.

Unfortunately, the natural deduction style is, although well-suited for use in the presentation of complete proofs, also ill-suited to your task of constructing proofs. An alternative style, called the *sequent calculus*, is appropriate if you want to prove something for yourself. MacLogic allows you to work in both styles: in either Check mode or Construct mode, corresponding respectively to the natural deduction style and the sequent calculus style. Check mode is appropriate if you want to check an existing natural deduction proof for correctness, and Construct mode is used if you want to construct a proof for yourself. In fact, the proof constructed is a sequent calculus proof: this can be translated mechanically, if you wish, by MacLogic, to a natural deduction proof, by the method outlined in section 4.3 below. The translation is fairly transparent.

1.3 Directions for use of manual

Novices should use this manual by skimming the above remarks, and working through the *Getting Started* section, referring whenever stuck to the on-line **Help** about the syntax, the rules, and the tactics. **Advanced users** will wish to explore the facilities for working in various logics, and should therefore read the section on *Advanced Work*. **Teachers** will wish to know how to install MacLogic, how to customise it for their own students, and how to construct problem files for their students to work on: they should read all sections of the manual, and the file “READ ME” accompanying the MacLogic application.

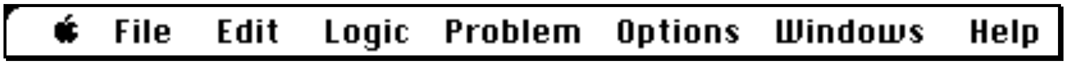
In case of difficulty, users of all kinds may need to ensure that MacLogic has been correctly installed.

2. Getting Started

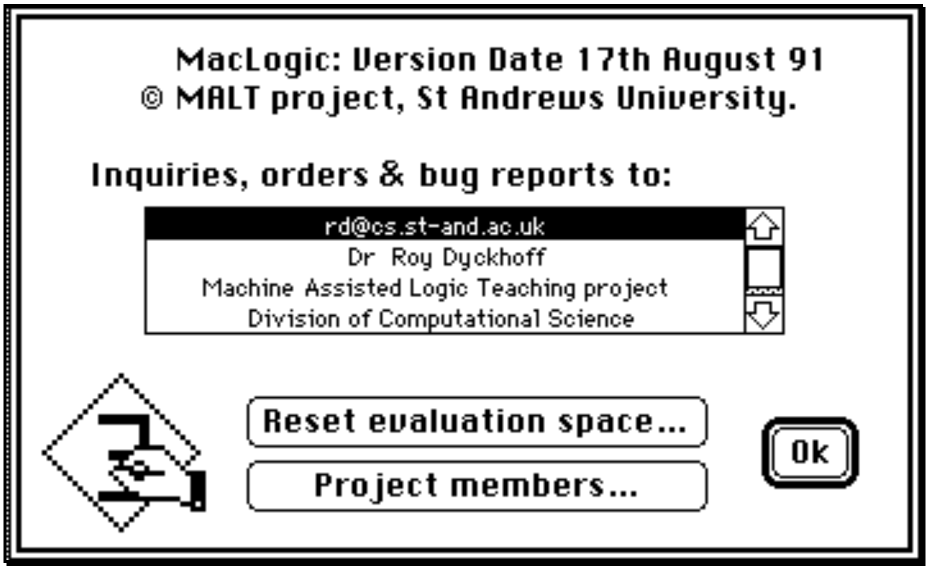
2.1 Checking a Proof

MacLogic is designed to help you both to check proofs and to construct them. There are two modes in MacLogic: *Check* mode, which allows you to check proofs for errors, and *Construct* mode, which helps you to build the proofs in a more organised manner. This section will take you through a sample session with the *Check* mode of MacLogic. You will meet *Construct* mode in section 2.2 below.

Suppose you are set the task of proving $P\to(Q\to R) \vdash (P\to Q)\to(P\to R)$. Open the MacLogic application by double clicking on the MacLogic icon. While it is loading, a small cursor containing the word MALT below a St Andrews cross is shown. After a little while[1], the menu bar



should appear, followed by a modal dialog box such as:



MacLogic: Version Date 17th August 91
© MALT project, St Andrews University.

Inquiries, orders & bug reports to:

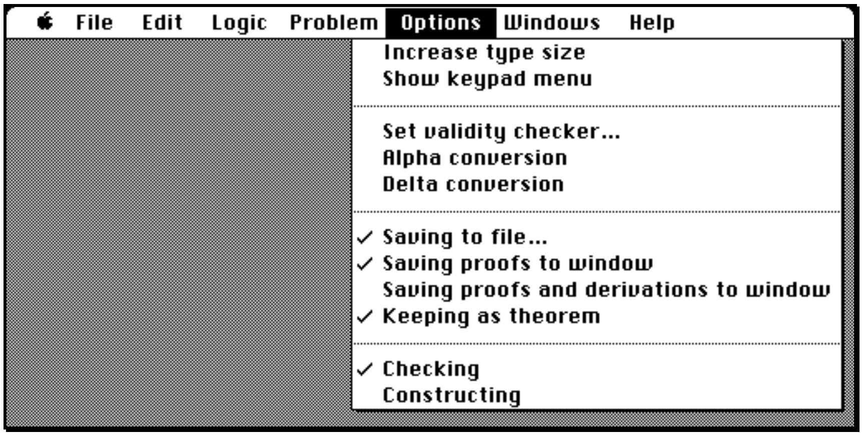
rd@cs.st-and.ac.uk
Dr Roy Dyckhoff
Machine Assisted Logic Teaching project
Division of Computational Science

You must respond to this dialog box before doing anything else. Just click on **Ok**, and respond to the next dialog box in the same way. Note that the latter is telling you that you can start work by using the **Problem** menu, and gives instructions on how to interrupt MacLogic if it gets stuck thinking about whether it can solve your problem by itself.

By clicking on the **Options** menu header, you can see whether *Check* mode is on by verifying that **Checking** is ticked. If it isn't, select it by dragging down till **Checking** is selected, then release the mouse button:

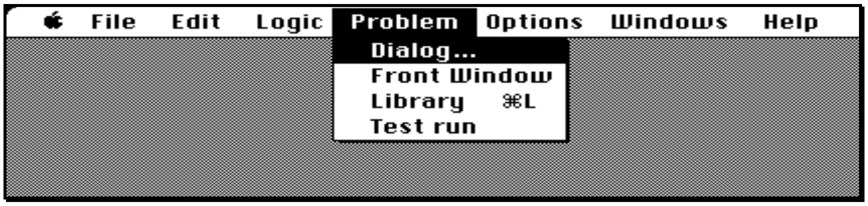
---

[1] Perhaps as much as 60 seconds on a MacPlus or Mac Classic.

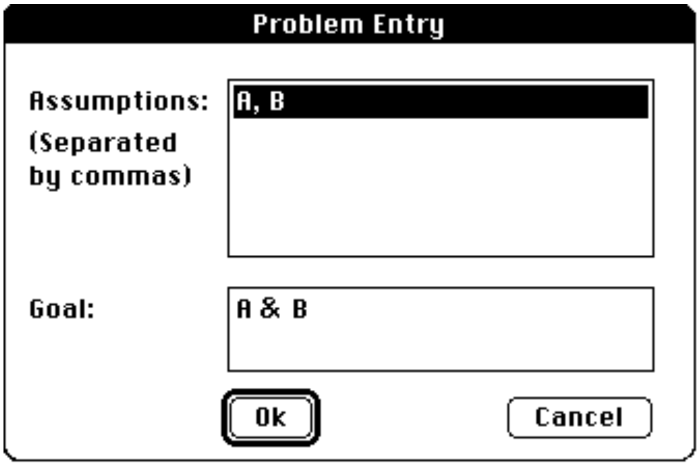


The **Logic** menu allows you to choose a particular logic. (MacLogic defaults to classical logic. You can explore the other logics later.) Note that in the **Options** menu, you can set the modes to save your proof to a file, to a window, or to keep the problem just solved as a theorem. Make sure all three of these are ticked, so you can see later what they do, that **Classical** is ticked in the **Logic** menu, and that all other items in that menu are unticked.

You can now start your proof by going into the **Problem** menu and selecting **Dialog…** : this is one of the ways of entering a problem. The other ways are to select a problem from the front window and to choose one from a problem library.



Once you have selected **Dialog…**, a modeless dialog box will appear looking like this:



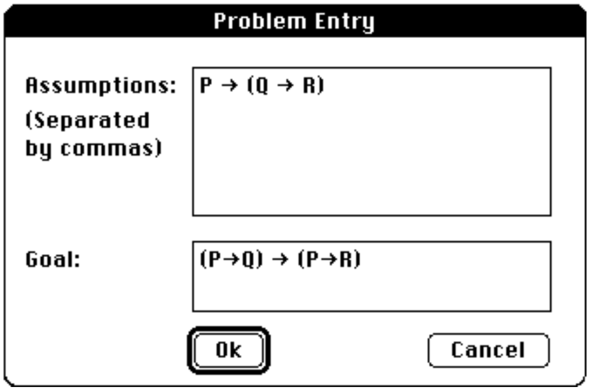
Problem Entry

Assumptions: (Separated by commas) A, B

Goal: A & B

This is the box into which you will type your problem. Note that it is prefilled with a very simple problem, just to show where things go: you will try something a bit harder! You can use the mouse or the tab button to move between the **Assumptions** field and the **Goal** field. If there is more than one assumption, they should be separated by commas, as in A, B.

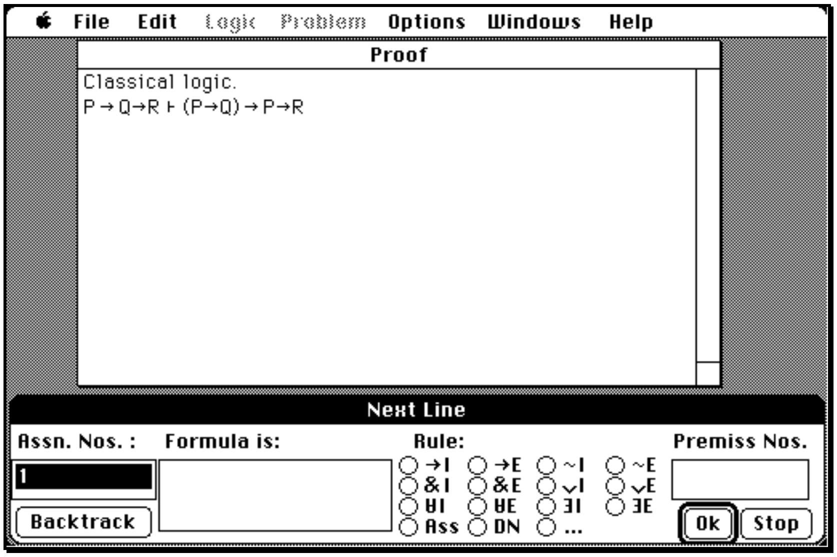
First, type P→(Q→R). (Note that → is, at least on British keyboards, typed as Shift Option Y.) Press the **Tab** key – that takes you to the **Goal** field: now type (P→Q)→(P→R). Once you have typed in the problem, the **Problem Entry** dialog box should look like this:



Assumptions: (Separated by commas) P → (Q → R)

Goal: (P→Q) → (P→R)

At this point, click on **Ok**. If you have not typed in a well-formed formula, you will be informed by an error message and must then correct your entries. You are now ready to begin the proof: the screen displays a **Proof** window, where your proof will be displayed, and a **Next Line** dialog box, into which you enter instructions for forming the next proof line. The screen will look[2] like this:

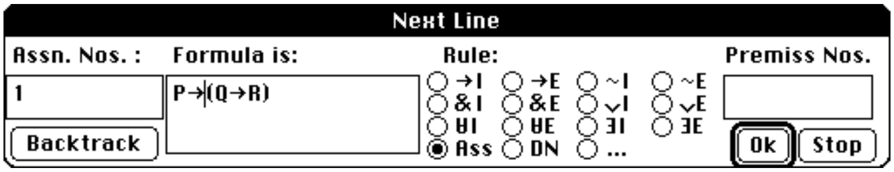


Classical logic.
P → Q→R ⊢ (P→Q)→P→R

Assn. Nos. : 1

Formula is:

Your first step is to put in any assumptions you might have. For the present problem, the only assumption is P→(Q→R), and so you fill in various parts of the **Next Line** dialog box like this:



Assn. Nos. : 1

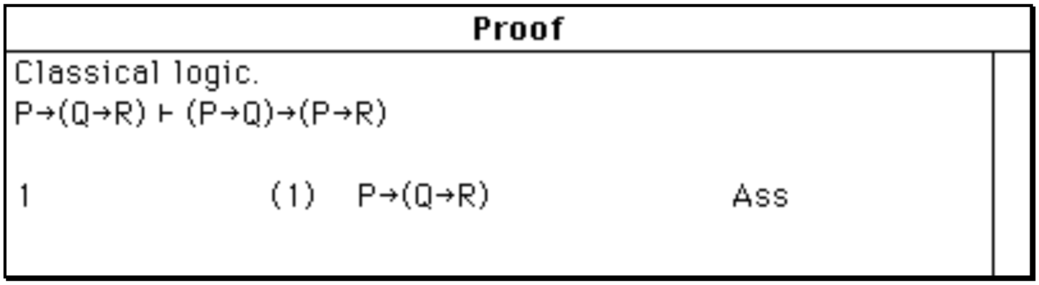
Formula is: P→(Q→R)

Rule: ● Ass

Remember you may use **Tab** to move from one field to another (you can also use the mouse, if you wish). Notice that because no premisses are needed for an assumption, the **Premiss Nos**. field is left blank. Once your **Next Line** dialog box has been completed in this fashion, click on **Ok**. The **Proof** window will change to look like this:

---

[2] The latest version of MacLogic has a **Next Line** dialog box differing in minor details from that illustrated: it is the name on the rule-button rather than its position which is important!

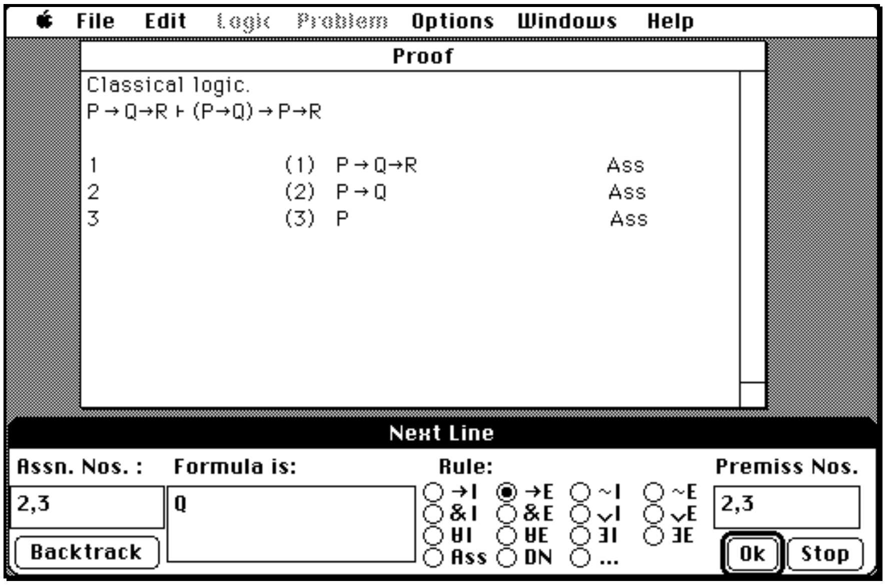


Classical logic.
P→(Q→R) ⊢ (P→Q)→(P→R)

1 (1) P→(Q→R) Ass

Your assumption line appears in the **Proof** window in a standard proof style (that of Lemmon), provided you have typed everything in correctly. If you haven't, an appropriate error message will appear and you must correct the fault. Notice that if your line is correct and it appears in the **Proof** window, the **Next Line** box is cleared, ready for the next line. Now you know the basics of entering assumptions. You will need two more assumptions (P→Q, P) for this proof – put them in now, one by one.

The next step after these assumptions is a use of the **→-Elimination** rule. It is abbreviated as **→E** in the **Next Line** box: all these buttons represent the rules which you can use. You will be learning all of them later: so far you should know **Ass**, **→E** and **→I** (**→-Introduction**). The **→E** step will look like this:



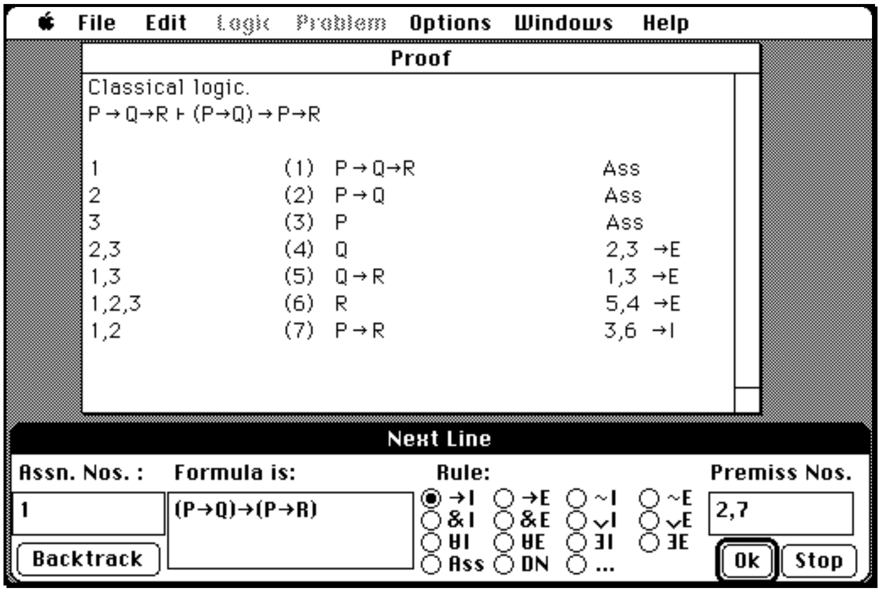
Classical logic.
P → Q→R ⊢ (P→Q) → P→R

1 (1) P → Q→R Ass
2 (2) P → Q Ass
3 (3) P Ass

Assn. Nos. : 2,3
Formula is: Q
Rule: →E
Premiss Nos. 2,3

You can see that you must now enter references to two premisses, as the **→-Elimination** step uses two previous lines as premisses. The premisses here are lines 2 and 3, and the new line depends on the assumptions made on lines 2 and 3. [Note that the premisses must be cited in order, with the major premiss, i.e. the line containing the formula with the '**→**' being "eliminated", first. Here this premiss is line 2. You can enter the assumption numbers, however, in any order.] Now click on **Ok**.

You can work through this proof line by line, entering them as you have been shown. You need only use **Ass**, **→E** and **→I**. The proof will continue to appear line by line in the **Proof** window. Once you have filled in the middle bit, you will find that entry of the last line looks like this:



```
Classical logic.
P → Q→R ⊢ (P→Q) → P→R

1          (1)  P → Q→R          Ass
2          (2)  P → Q            Ass
3          (3)  P                Ass
2,3        (4)  Q                2,3  →E
1,3        (5)  Q → R            1,3  →E
1,2,3      (6)  R                5,4  →E
1,2        (7)  P → R            3,6  →I
```

Assn. Nos. : 1

Formula is: (P→Q)→(P→R)

Rule: ● →I ○ →E ○ ~I ○ ~E ○ &I ○ &E ○ ∨I ○ ∨E ○ ∀I ○ ∀E ○ ∃I ○ ∃E ○ Ass ○ DN ○ ...

Premiss Nos. 2,7

When you now press **Ok**, the proof will be finished and you will be congratulated with a “Well done!” message. Your completed proof looks like that illustrated on the first page of this manual.

Should you ever want to go back and see the proofs you’ve done in the present session, go to the **Windows** menu and select **Previous Proofs**. You will see the proof you have just completed, as well as any others you may have saved to the window in this session.

You may print this proof; to do so, select **Print visible windows...** from the **File** menu: this will print all the windows presently on the screen. Remember first to hide any windows on the screen which you don’t want to print, by clicking in their “close boxes”.

If you make a mistake, note that you can undo it: just click on the **Backtrack** button in the **Next Line** dialog. Repeated clicking on this undoes the lines, one at a time, until you are back at the start. Note that you can use the **Edit** menu to copy formulae from the **Proof** window into the dialog box. You can’t paste them into the **Proof** window, lest you construct incorrect proofs! Try all items on the **Edit** menu!

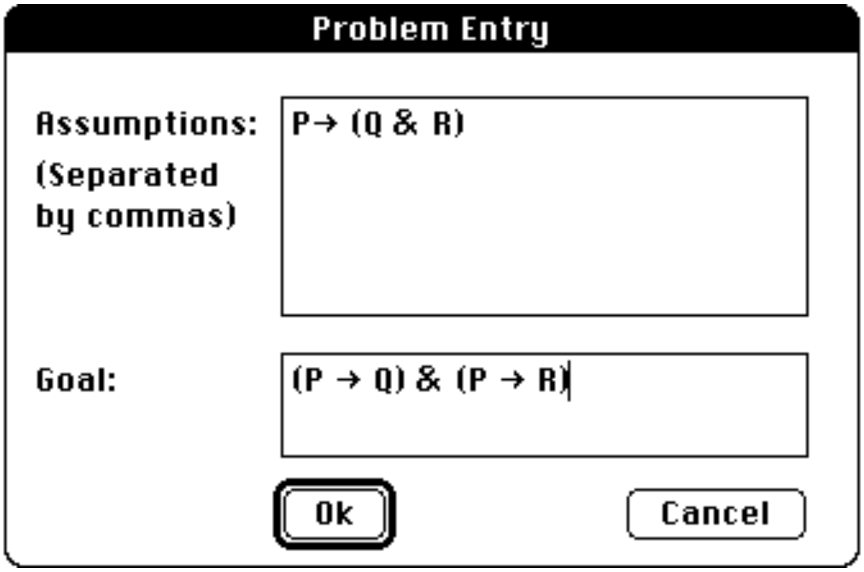
If you get tired of doing a proof, you can abandon it by clicking the **Stop** button. You will be asked to confirm your decision. Any work you have done on the proof will be lost - but you can always copy the text in the **Proof** window to a window such as the **Jotter** to remind yourself what steps you took.

2.2 Constructing a Proof

Once you are acquainted with the Check mode of MacLogic, the next matter to tackle is Construct mode: this mode allows you actually to build a proof (more easily), and not simply to verify that a proof is correct. It works in what is called a *top-down* fashion; that is to say, it doesn't work forward from the assumptions to the conclusion in the way that we have done so far when working in Check mode, but requires you to consider the problem as a whole and use some more strategic thinking. Sometimes you will (in effect) work down from the assumptions, but you can also work backwards from the conclusion—in general, tactics for elimination rules work down from the assumptions, and tactics for introduction rules are used to work up from the conclusion.

Because of this difference in approach between Check and Construct mode, you will also notice a considerable difference in the style in which your work appears on your screen. To emphasize this difference, we put the record of what you do into a window called **Derivation**: when all the work is complete, this is transformed in a mechanical fashion (see section 4.3 for details) into a proof in the traditional style. You start with a *problem* consisting of (optional) assumptions leading to a conclusion, and working on this problem produces a series of sub-problems as you go along, which you must solve one by one.

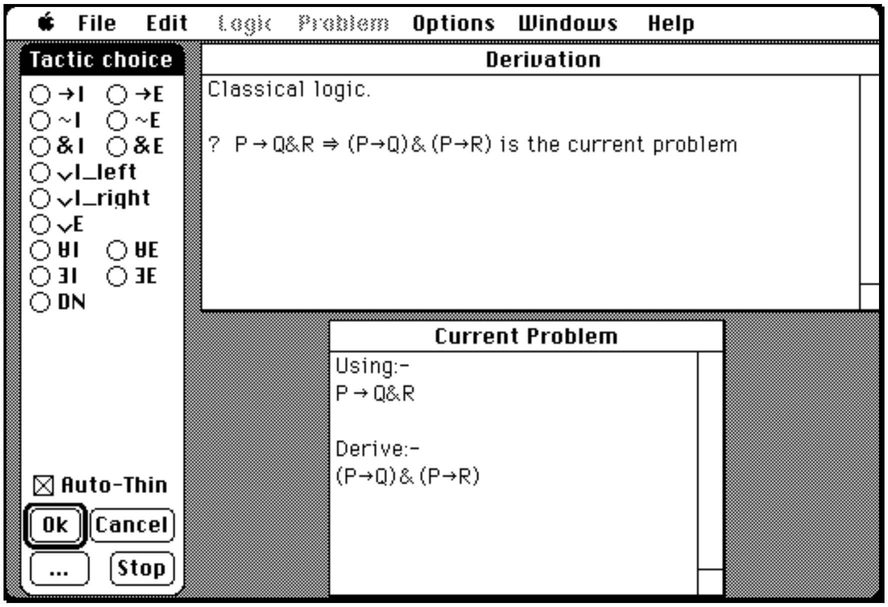
Suppose we are set the task of proving $P \rightarrow (Q \& R) \vdash (P \rightarrow Q) \& (P \rightarrow R)$. As you have done before, double-click on the MacLogic icon (unless it is already running), and select the item **Constructing** in the **Options** menu. Select **Dialog…** from the **Problem** menu, and fill in the **Problem Entry** box as follows:



Assumptions: P→ (Q & R)

Goal: (P → Q) & (P → R)

Click on **Ok**, and you will find that a window called **Derivation**, another called **Current Problem**, and a dialog box called **Tactic choice** will appear. **Derivation** will display all the problems and sub-problems which you are to solve, and **Tactic choice** has a selection of buttons for tactics which you use to solve the current problem. (For a more complete description of what all these buttons do, see section 5.4.) The screen will look like this:



Classical logic.

? P → Q&R ⇒ (P→Q)&(P→R) is the current problem

Using:-
P → Q&R

Derive:-
(P→Q)&(P→R)

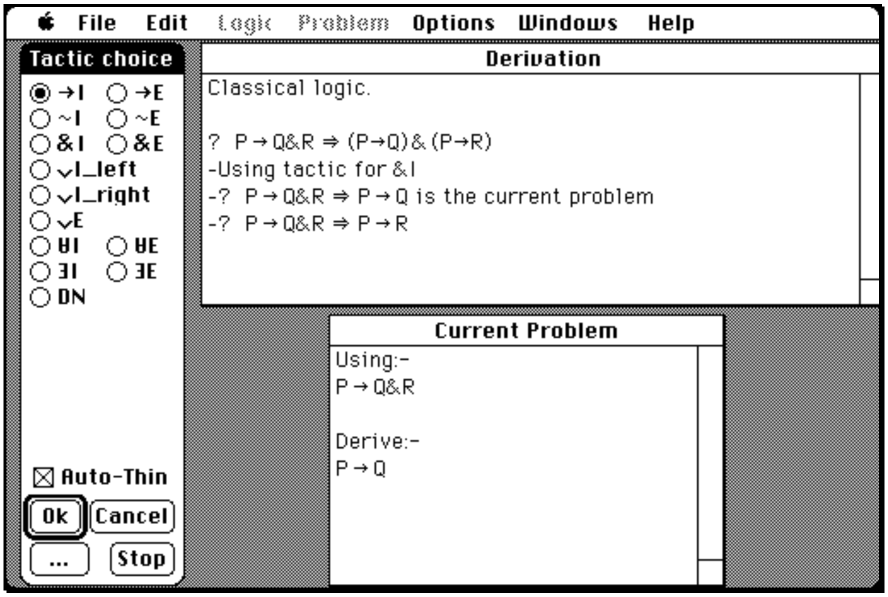
Note that the antecedent formula $P \to Q\&R$ appears without parentheses: there are some conventions (see section 5.2 below) about when parentheses can be omitted. Some small spaces are included (in this case on each side of the '→' symbol, telling you discreetly which operator is the principal operator of the formula.

Note also that you will always be told, in the **Current Problem** window, what the current problem (the sub-problem that you are working on) is. It is just a restatement of information in the **Derivation** window, in a more familiar layout style. Also, note the indentation of the current line, in the **Derivation** window: it will always show what level of sub-problem you are at. This should become clearer as you work through the proof.

Your first step is to choose a tactic. You will notice that there is much less typing involved here than in the **Check** mode. This is one of the reasons why **Construct** mode is far better for constructing proofs; also, you needn't worry about having explicitly to make assumptions; because of the way in which **Construct** mode works, they are made automatically for you, when necessary.

The natural place to start in this problem is with an **&-Introduction** step — because it is an introduction rule, it will work backwards on the conclusion. The current problem displayed is to derive a conjunctive formula $(P \to Q) \& (P \to R)$ from $P \to Q\&R$. So the appropriate choice is to use the tactic for **&I** to obtain it. The goal formula $(P \to Q) \& (P \to R)$ will be split up into the two conjuncts, and you will then have two major sub-problems to solve. Each of the sub-problems will have the same fact-list. We call the list of formulae (so far there is only one) to the left of ⇒, the *fact-list*, for it consists of "facts" from which we are trying to prove the goal. The "facts" consist of assumptions plus what we derive from assumptions in elimination rules on our way to achieving the goal. We also call this list the *antecedent* of the problem.

The **&I** button is selected as shown, and confirmed by pressing **Ok**. You now have two sub-problems, with the same antecedent, arising from the two conjuncts, and we are set to work on the first, *viz* $P \to Q\&R \;\Rightarrow\; P \to Q$. (You will tackle the second, *viz* $P \to Q\&R \;\Rightarrow\; P \to R$, only after you have solved the first.). To solve the first, use an **→I** step:

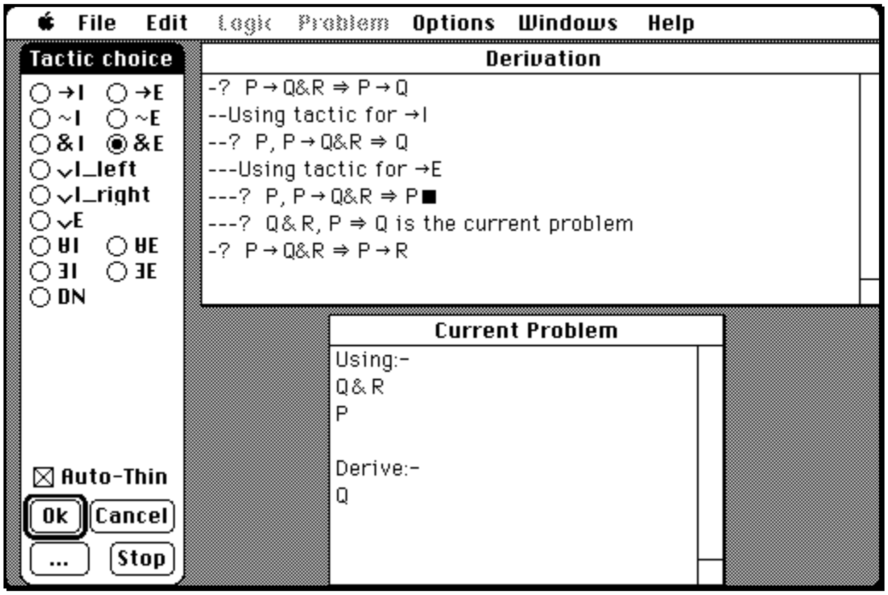


```
Classical logic.

? P→Q&R ⇒ (P→Q)&(P→R)
-Using tactic for &I
-? P→Q&R ⇒ P→Q is the current problem
-? P→Q&R ⇒ P→R
```

```
Using:-
P→Q&R

Derive:-
P→Q
```

This will bring the formula P back from the goal, P→Q, into the antecedent as an assumption:

Notice that we are still working on the first major sub-problem, and that our second major sub-problem remains to be solved ahead. Now, if we examine the formulae in the antecedent, we can see that an **→E** step is in order. By clicking on **→E** and then on **Ok**, we get the following screen:

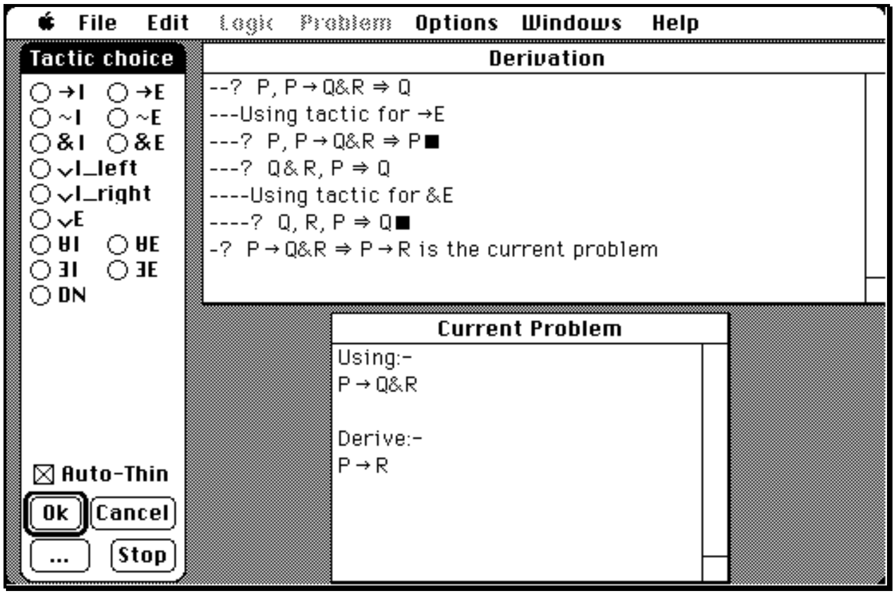


```
-? P→Q&R ⇒ P→Q
--Using tactic for →I
--? P, P→Q&R ⇒ Q
---Using tactic for →E
---? P, P→Q&R ⇒ P■
---? Q&R, P ⇒ Q is the current problem
-? P→Q&R ⇒ P→R
```

```
Using:-
Q&R
P

Derive:-
Q
```

The black box ■ indicates that you have just solved a trivial sub-problem: it is trivial because the formula, P, to be derived appears as one of the facts (actually, as one of the assumptions).

The next sub-problem consists in deriving Q from Q&R and P. For the first time we have a formula in our fact-list which is not an assumption, but is derived from the assumption P→(Q&R) by →E. This sub-problem is easily solved, using an **&-Elimination** step to split up the conjunction Q&R: this will leave us with Q both in the antecedent (on the left of the arrow) and as a goal (on the right): this sub-problem is now trivial.

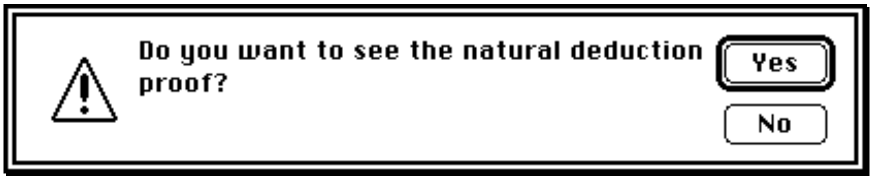


```
--?  P, P → Q&R ⇒ Q
---Using tactic for →E
---?  P, P → Q&R ⇒ P■
---?  Q& R, P ⇒ Q
----Using tactic for &E
----?  Q, R, P ⇒ Q■
-?  P → Q&R ⇒ P → R is the current problem
```

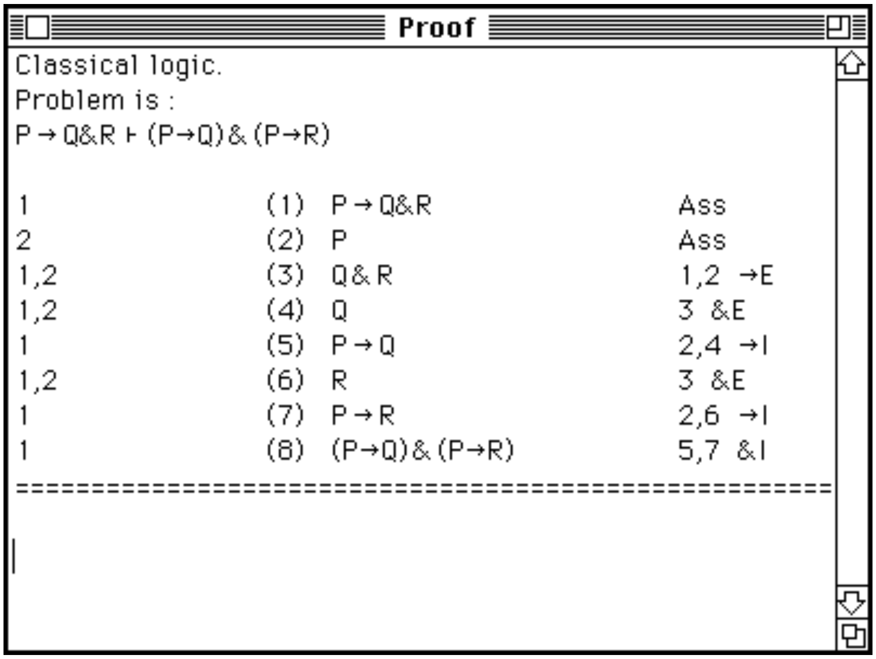
```
Using:-
P → Q&R

Derive:-
P → R
```

We have now solved one of the two major sub-problems that we started out with (as indicated by the indentation on the left). The next sub-problem is almost of the same form, and is solved using the same steps of **→I**, **→E** and **&E.** Try this now.

Once you have successfully completed these steps, you will be rewarded with a 'Well done!' message, as in the Check mode. You will then be asked if you wish to see the derivation transformed into a natural deduction proof:


Do you want to see the natural deduction proof?

Select **Yes** and the derivation will be mechanically transformed into a Lemmon-style natural deduction proof, displayed in the Proof window:



```
Classical logic.
Problem is :
P → Q&R ⊢ (P→Q)&(P→R)

1               (1)  P → Q&R            Ass
2               (2)  P                  Ass
1,2             (3)  Q&R                1,2  →E
1,2             (4)  Q                  3  &E
1               (5)  P → Q              2,4  →I
1,2             (6)  R                  3  &E
1               (7)  P → R              2,6  →I
1               (8)  (P→Q)&(P→R)        5,7  &I
=======================================================
```

You can now go back and examine the strategy you used in the **Derivation** window, comparing it with the proof. As in Check mode, printing may be done using the **Print visible windows…** item from the **File** menu. You can print this proof now, or keep it in the file of proofs for printing later, as before.

3. Advanced Work

3.1 First-order logic and equality

The syntax of first-order formulae (in MacLogic) allows the use of variables, a ... z, as terms. It does NOT allow compound terms such as x+y or f(x,y). There is no syntactic distinction between free and bound variables, nor any division into “constants” and “variables”. No parentheses are needed between predicate letters, A ... Z, and their arguments.

Free and bound occurrences of variables are defined as usual - see the file “Syntax Help” for formal details. Vacuous quantification and repeated quantification are prohibited.

Standard inference rules, for introduction and elimination of the universal and existential quantifier, are provided.

When using inference rules and tactics, it is important to know when two formulae are identical. (For example, *modus ponens*, the rule of →-Elimination, operates on two formulae A → C and B, and is acceptable provided the formulae A and B are the “same”.) We therefore have a precisely defined notion of *identity* between formulae. This is routine for zero-order formulae: but it becomes non-trivial when bound variables are present.

Two conventions are possible. One is that two formulae are identical if (ignoring spaces and superfluous parentheses) exactly the same characters are used in each, in the same order. The other is that two formulae are identical if they are what is called, in the lambda calculus, *alpha-convertible*, i.e. they differ only in the names used for bound variables. Thus, according to the first convention, the two formulae (∀x)(∀y)Awxy and (∀y)(∀z)Awyz are not identical, whereas according to the second, they are identical. Note that neither is identical, under either convention, to (∀w)(∀x)Awwx, since w in the latter has no free occurrence.

Both conventions are encoded in MacLogic: the choice between them is made from the **Options** menu, by ticking or unticking the **Alpha conversion** item. (Note that the notion of identity actually includes the possibility of expanding definitions as well - see 3.3 below.)

This has the effect that if alpha conversion is on, the problem of deriving, say, (∀x)(∀y)Awxy from (∀y)(∀z)Awyz is trivial, whereas if it is off, one has to do two ∀-elimination steps followed by two ∀-introduction steps.

The usual restrictions on variables for the quantifier rules are implemented. In Construct mode, the choice of the tactic for ∃-elimination or ∀-introduction requires the use of a new variable, not occurring free in the current problem. MacLogic either chooses a new variable for you, or asks you to choose one from a list, according to whether alpha conversion is on or off. Likewise, the other quantifier tactics require you to enter a term (actually just a variable, since we have no compound terms or constants). If alpha conversion is off, this must be free for the bound variable being replaced, and is rejected if not; if alpha conversion is on, any variable is accepted, and the bound variables are renamed to avoid capturing the variable in the entered term.

One feature, in Construct mode, of the tactics for quantifiers is a little odd: suppose one is trying to solve the problem

$$(\forall x)Ax \Rightarrow (\forall y)Ay$$

with alpha conversion off. One obvious way is to replace it by the problem

$$(\forall x)Ax \Rightarrow Ay$$

and now use the tactic for ∀-elimination, choosing the term y to instantiate the bound occurrence of x: this is transformed to the natural deduction proof

1	(1)	(∀x)Ax	Ass.
1	(2)	Ay	1 ∀E
1	(3)	(∀y)Ay	2 ∀I

One might also wish to be able to replace the original problem by

$$Ay \Rightarrow (\forall y)Ay$$

and now use the tactic for ∀-introduction. This is not allowed, since the new variable y used to replace the bound occurrence of y on the RHS occurs free in Ay. In fact, it does not occur free in the assumption (∀x)Ax from which Ay was derived, so the use of y is harmless: nevertheless, the sequent calculus rules forbid use of y as a new variable at this point. This restriction does not however reduce the range of natural deduction proofs which can be constructed.

Equality may be selected from the **Logic** menu, in which case the tactics and rules for equality may be used. (Equations are always acceptable as formulae, even if **Equality** is not selected in the **Logic** menu.) The rules for equality are accessible (in the **Next Line** dialog box) from the **=...** radio button (there isn't room for them otherwise!) The tactics for equality appear as radio buttons in the **Tactic choice** dialog box.

Equality (often, but not in MacLogic, called "identity") is an equivalence relation, i.e. is reflexive (everything is equal to itself), symmetric (one thing equals another if the other is equal to the first), and transitive (whatever equals something equal to a third thing is itself equal to the third thing). But equality is in a sense the smallest equivalence relation, so if a and b are equal, then anything true of a is also true of b - a congruence principle incorporated as the rule **=E**, in counterpoint to the reflexivity incorporated as the rule **=I**. [The symmetry and transitivity can actually be derived from these two rules, but it is convenient to make them available as if they were primitive rules of the logic.]

Using the rules for equality (in MacLogic's Check mode) is straightforward. The tactics for equality (in Construct mode) are straightforward in the case of reflexivity (in fact, any goal of the form x = x is regarded as trivial, and you are not required to invoke the tactic for **=I**), symmetry, and transitivity. That for **=E** requires more care. The tactic is that if you know an equation $s = t$, and the goal is any formula C, you may replace the current problem by two sub-problems - the first of showing $A(s)$, and the second of showing C from the additional knowledge of $A(t)$, where $A(v)$ is a formula with one or more occurrences of a free variable v. (If there are no such occurrences, the tactic reduces to the **Cut** tactic.) You are required to enter the formula $A(v)$, and to indicate what variable v is to be substituted for, by means of a dialog.

For example, suppose we want to prove that = is symmetric, without using the **=symm** tactic. Our initial problem (we show formulae parenthesised for clarity) then is, after removal of quantifiers and use of the tactic for **→I**:

$$(x = y) \Rightarrow (y = x)$$

We now use the tactic for **=E**, and enter the formula $A(v) =_{def} (v = x)$, with v as designated variable. (What you actually type in is `v = x` in one edit field, and the variable `v` in the other.) The new sub-problems are first to show $A(x)$ from $(x=y)$, and secondly to show $(y=x)$ from $A(y)$ and $(x=y)$, i.e.:

$$(x = y) \Rightarrow x = x$$

$$(y = x), (x = y) \Rightarrow (y = x)$$

both of which are trivial.

3.2 Modal logic

Any of MacLogic's basic logics - classical, intuitionistic and minimal - may be extended by rules for modal operators. The modal operators are '□', where □A means "necessarily A", '◇' where ◇A means "possibly A", and '⥽' where A⥽B means "A strictly implies B". MacLogic treats all three as primitive, although when using a classical base (though not an intuitionistic or modal one) some authors use '□' to define the others, since classically ⊢ (◇A ↔ ~□~A) and universally ⊢ ((A⥽B) ↔ □(A→B)).

There are many systems of modal logic, characterising different notions and strengths of modality. MacLogic implements the two most famous: Lewis' systems S4 and S5.

The □-Elimination rule is the same in both systems: □E says that A may be inferred from □A, the assumptions staying the same. Similarly, the rule ◇I allows ◇A to be inferred from A, and ⥽E allows B to be inferred from A⥽B and A (pooling the assumptions). But when may □A be inferred from A? Answer: when the assumptions on which A depends are suitably strong. □I has the same form in both systems, to infer □A from A, the difference lying in the restriction that the assumptions on which A depends are appropriately modal — S4-modal in the one case, S5-modal in the other. S4- and S5-modal are defined in Section 5.2. ⥽I makes similar use of the notion 'modal', in line with A ⥽ B's possible definition as □(A → B), while ◇E uses in addition a notion of comodality, also defined in Section 5.2.

The rules for '□' and '◇' may usefully be compared with those for '∀' and '∃' respectively. For, semantically, □A means that "A is true in all (accessible) possible worlds", and ◇A means "A is true in some (accessible) possible world".

For further details of these modal logics, see the books (referenced in the bibliography, section 4.5), by Lewis & Langford, Hughes & Cresswell, Zeman and Prawitz.

3.3 Definition expansion

Definitions are a useful way of increasing the expressive power of a language. If one can define a new concept in terms of old ones, then all one's knowledge about the old ones can be applied to the new concept.

In MacLogic, definitions are used rather sparsely. Definitions of two logical symbols are available: the definition of negation in terms of implication and contradiction, and of equivalence in terms of implication and conjunction, detailed below.

One can choose whether or not the definition is applied automatically by MacLogic. One point of view is that if two formulae are "definitionally equal", as are for example $\sim A \; \& \; B$ and $(A \rightarrow \Lambda) \; \& \; B$, then the use of one in a context where the other should be used should be acceptable without comment. The other point of view is that the replacement of a formula by a definitionally equal formula should involve invocation of the rule "Definition".

The two views are reflected by adjustments to the item **Delta conversion** in the **Options** menu. If Delta conversion is ticked, then two definitionally equal formulae can be used interchangeably without comment. If Delta conversion is unticked, then expansion or contraction of formulae using definitions must be explicitly invoked. We now describe how to do this just in the case where Delta conversion is unticked. [When it is ticked, there is neither necessity nor possibility of invoking the definition mechanism.]

In Check mode, suppose we have established a formula P, depending on certain assumptions. We may now derive a formula Q, depending on the same assumptions, by invoking the rule "Defn.", accessed via the … radio button, and quoting as premiss the line number on which P was established, *provided that* P and Q are definitionally equal. [Note that if P and Q are actually the same formula, this can still be done, but it may look a little odd!]

In Construct mode, suppose we are working on a problem $F \Rightarrow A$, where F is a formula list and A is a formula. If we invoke the tactic for "Definition", accessed via the … radio button, MacLogic looks to see whether or not any of the formulae in the problem have a principal symbol which is a defined symbol, and thus admit a rather simple expansion. If there is only one such formula, the expansion is done immediately, the move from one line to the next being marked by the comment "Using tactic for Definition". If there are several such formulae, you are asked to choose which one is to be expanded.

Note that there is a restriction here: in Construct mode, you may not (in effect) point at a formula and say that it is to be replaced by a definitionally equal formula: all you can do is expand a definition used 'at the top level'. This avoids the necessity of typing in the new formula.

For example, $P\leftrightarrow Q$ can be expanded to $(P\rightarrow Q)\&(Q\rightarrow P)$, since $\leftrightarrow$ is a defined symbol; but $(P\leftrightarrow Q)\&R$ cannot be expanded to $((P\rightarrow Q)\&(Q\rightarrow P))\&R$, even though they are definitionally equal.

Note also that if **Delta conversion** is ticked, you may use the rules for implication in place of the rules for negation: negation is just a special case of implication. Likewise, equivalences $A \leftrightarrow B$ are just a special case of conjunctions, so the rules for $\&$ are applicable. But when **Delta conversion** is not ticked, then a formula of the form $A \leftrightarrow B$ must be expanded explicitly, as described above, before its components can be used.

The definitions currently built into MacLogic are that

$\sim A \quad =_{\text{def}} \quad A \rightarrow \Lambda.$

$A \leftrightarrow B \quad =_{\text{def}} \quad (A \rightarrow B)\ \&\ (B \rightarrow A).$

where A, B range over arbitrary formulae. [$=_{\text{def}}$ is a symbol traditionally used between a *definiendum*, on the left, and the *definiens*, on the right.]

3.4 Sequent Introduction

In practice, few proofs are done just by means of the primitive rules of inference corresponding to the logical constants: one appeals to previously proved results. MacLogic provides a mechanism for making such an appeal. It is limited to the use of results from zero-order logic.

For example, suppose you are using classical logic, and have already shown that $A \vee {\sim}A$ is provable therein: and then you have a problem of deriving C from the two implications $B \rightarrow C$, ${\sim}B \rightarrow C$. You can proceed to use the primitive rules, but it is more natural to quote the result $B \vee {\sim}B$, and then use $\vee$-Elimination. This step of quoting a(n instance of) a previously proved result is called[3] “Sequent Introduction” (**S.I.**).

S.I. can only be used if some theorems are in the database, having been proved already, either in the same session (and kept) or in a previous session (and loaded). If you try to use it when no theorems are in the database, you have the opportunity to load some. The mode of use varies according to which of Check mode and Construct mode you are in.

In Check mode, **S.I.** is accessed via the **...** radio button on the **Next Line** dialog box. Before using this button you may wish to look at the **Theorems** window to see which theorem you are going to use. There is then a dialog, asking you to choose a theorem. The premisses cited by number in the **Premiss Nos.** field must be, in order, the (substitution instances of the) assumptions in the theorem being used: the formula in the **Formula** field must be the appropriate instance of the consequent of this theorem. The assumptions listed in your **Assumption Nos.** box must be all the assumptions needed in all the premisses to which you refer.

In Construct mode, suppose your current problem is to prove C from a list F of formulae, and that a problem $G \Rightarrow D$ has already been solved and kept as a theorem. Then you can construct a substitution instance $G' \Rightarrow D'$ of this theorem, and then try and solve the new sub-problems of trying to show, from F, each in turn of the assumptions in G', and then, adding D' to F, of trying to show C. In many cases, the theorem is chosen to make most of these steps trivial, and one might think the tactic could be more simple to use: but for completeness, this rather general form is required, alas. The tactic for **S.I.** is accessed via the **...** button.

Use of the **S.I.** rule thus gives you the right to use, very much as if it were an extra rule of inference, any previously proved zero-order result. (Not all correct rules can be represented in this way - for example, those involving discharge of assumptions cannot be so represented.)

3.5 Use of tautologies

Rather as in the above explanation about the **S.I.** rule, there is another method for avoiding unnecessary work: the appeal to a problem’s being obviously solvable. For example, when one is learning the rules and tactics of first-order logic, it is irritating to have to work with the zero-order rules rather than concentrating on the first-order difficulties, so one would like to dismiss the zero-order sub-problems as trivial. One may of course be able to quote an earlier result, but one’s library of theorems is usually

---

[3] So it is called in the book by Lemmon, who distinguishes between sequents in general and those, which have no assumptions, which he calls "theorems". For us, theorems are provable sequents, and are of two kinds - "categorical", having no assumptions, and "hypothetical", being based on assumptions. What he calls "Sequent Introduction" would make more sense if renamed "Theorem Introduction", except that might confuse devotees of Lemmon's book.

deficient. One therefore often just uses the rule or tactic labelled[4] "Tautology", and relies on the theorem-provers built into MacLogic to check that this is justified.

To use the **Tautology** rule in Check mode, use the … button to get at it. The provability (in zero-order logic) of your conclusion (in the Formula box) from the premisses cited is checked by the theorem-prover. The assumptions quoted should be all the assumptions on which any of the premisses depends. See below for details of what the theorem-prover can do: it is good at problems in various first-order logics, but not allowing the rules for equality or modal operators.

To use the tactic for **Tautology** in Construct mode, use the … button to get at it. You must then enter the precise sequent being appealed to in a dialog box. (The box may be pre-filled with a suggestion.) This is checked[5] by the theorem-prover for being provable in zero-order logic: if it is accepted, its assumptions are taken, one by one, as new goals in the current context, and its conclusion added to the current context for proving the old goal. Derivations built in this way are transformed to proofs using the **Tautology** rule described above.

---

[4] Tautology is a misnomer, for two reasons: traditionally, a tautology is just a formula, rather than a sequent, and it should be provable by zero-order rules only. The name is also rarely used except in classical logic, since the usual definition is based on truth-tables. We use the name until we can think of a better substitute, capturing the notion that we regard the current sub-problem as trivial.

[5] Similar remarks to those just given for Check mode describe the limitations of the theorem-provers.

3.6 Cut rule

One important property of many logical systems is that a certain rule, called the “Cut” rule, is admissible, in the sense that its addition to the system does not extend the range of theorems that can be proved, but just allows the proofs to be shorter or simpler. Gentzen’s “Hauptsatz”, or “Cut elimination theorem”, says just this about certain sequent calculus formulations of classical or intuitionistic logic.

In the search for a proof of a sequent, one usually relies on just looking at the formulae involved, breaking them up syntactically: the tactic corresponding to the Cut rule, however, requires one to have insight by picking an appropriate formula, the “Cut formula”, out of thin air. This rule’s presence makes the automatic search for proofs rather hard - so it is better if it is not a necessary part of the formal system. MacLogic is therefore designed to be used without a Cut rule.

However, no convenient formulation of the modal logic S5 is known to be cut-free. It is therefore necessary to be able to use the Cut rule (or, rather, its corresponding tactic), when working in S5. There is therefore a tactic for Cut, accessible via the **…** radio button from the **Tactic choice** dialog.

Technically, the Cut rule is a rule of the sequent calculus. Its use translates into the use of a natural deduction rule which we call "Substitution", since it corresponds to the substitution of a proof of the cut-formula A in place of an assumption of A. (The notion of a “Cut” can be defined for natural deduction, and one can then talk about “Cut-elimination”; we prefer to talk of “proof normalisation”.)

3.7 The window system

MacLogic includes a conventional window system, allowing the creation, opening, saving, and killing of windows. Some windows, like **Derivation** and **Proof**, are created for you: their contents can only be changed by using the formal systems, to ensure the logical correctness of their contents. Other windows (from the **Jotter** downwards in the **Windows** menu) can be edited freely: you can use them if you like for creating proofs, but MacLogic will not guarantee the correctness of proofs therein.

Window names should be sensible, and may not conflict with file names: i.e. if you create a file it should not be given the same name as a window. (The file and window input/output work by reading to or from a channel, identified just by its name, hence obvious problems if names clash.)

Windows have “close boxes”, at the top left corner: but they continue to be in memory after being closed, and can be re-opened again, usually by selection of the window’s name in the **Windows** menu. Certain windows are part of the help system, and are only opened by selection of appropriate **Help** commands.

Text windows always appear in the Konstanz font. You can switch the size of the font between 9pt and 12 pt.

If you are a teacher using an LCD panel to show MacLogic to a class, you may find it convenient to put certain windows in bold face: use the **Windows** menu item **Toggle bold**.

The help system works by using five files [“ATP Help”, “Syntax Help”, “Menu Help”, “Tactics Help” and “Rules Help”]. The first two are displayed directly; the last three are invisible, but extracts from them are displayed on request. Each of the files is loaded into memory when first needed; they are all killed if you invoke the command **Help/Memory problems** (of course, they can be loaded again if you wish).

Note that the dialogs are actually windows, of a different type: so, some of the commands that apply to windows, like editing commands, can be used on dialogs as well. One non-obvious feature of dialogs is that if you move a dialog, it should appear in the same place on the screen next time it is used - handy if you have a large screen. (Some dialogs, the so-called ‘modal dialogs’ – don’t confuse this sense of ‘modal’ with ‘modal logic’ – cannot be moved: they must be answered immediately.)

3.8 Using the validity checker

There is a window of information about the “validity checker” (*alias* “theorem prover”) accessible on-line, via **Help/ATP**. The data in it is loaded from the file “ATP Help”. In brief, MacLogic’s validity checker is intended to operate quietly in the background when you generate a new problem, and to warn you if it can’t solve the problem itself. It is believed to be sound, and to be complete for the problems solvable by zero-order non-modal rules.

The validity checker can be set (via the **Options** menu) to consider all problems, just the quantifier-free problems or no problems at all. It will ignore the modal rules and the rules for equality - taking them into account would slow it down to an unacceptable speed. There is a parameter which you can adjust to control the complexity of search: set at level 0, it will tackle[6] all zero-order problems, and some with quantifiers. Level 1 is adequate for a wide range of first-order problems, but is generally slower. Level 2 is needed for some deceptively simple problems like (in Classical Logic)

$$\Rightarrow \quad (\exists x)(\forall y)(Fx \rightarrow Fy)$$

and should normally be adequate for any problems attempted by your students. Levels 3 to 7 should rarely be needed.

The validity checker can be used in two ways. First, it can monitor your work: when you start to tackle a problem, or one of its sub-problems, it can warn you about the problem’s unsolvability. Second, you can invoke it directly from the **Help/Valid** menu item: for example, if you want to have a theorem proved automatically for you and added to the database, or if you are working on a problem and want to see if MacLogic can solve a similar problem, perhaps in a weaker logic. (Note that if you invoke the **Help/Valid** menu item while a process such as the construction of a proof is suspended, only a modest amount of memory is available for this second process, because of the way MacPROLOG organises its memory.)

When you use the **Help/Valid** command, the settings of the validity checker can be adjusted for the duration of MacLogic’s work on the problem you enter—but this doesn’t affect the settings used the rest of the time. To adjust those, use the command in the **Options** menu.

Remember that if you want to keep the theorems that you (or MacLogic) prove(s), ensure that **Keeping as theorem** is ticked in the **Options** menu.

See section 5.1.3.4 below for details of how to add lots of theorems quickly to the database.

---

[6] Of course, there are well-known results on the complexity of theorem proving even for zero-order logic, so this claim is optimistic. What is meant is that on a large enough machine with a fast enough processor, it will eventually report either “Valid” or “Invalid”. In contrast, first-order problems may cause any sound and complete theorem prover, however clever, to run for ever.

4. Theoretical background and biblio-graphy

4.1 Introduction

It is of course not at all necessary to understand all the theory behind MacLogic to be able to use it effectively. But the interested student may (and the teacher must) enquire as to why it is as it is, and how it works. In this section we try to answer some of these questions.

First, note that with the availability of computers it is now feasible to mechanise most aspects of the tedious task of checking proofs for correctness. But one may also implement the methods (the *algorithms*) for constructing proofs, and then reflect on how much one wants to take away from the student the task of doing this for herself. Of course, one really wants to provide a variety of levels of help - from the completely automatic checking that a problem is solvable to sensible assistance to the student discovering her own proof. The computer's behaviour should ideally adapt to different levels of expertise: MacLogic, however, is adaptable rather than self-adapting.

Second, the use of computers has led to a re-evaluation of the use of different formal systems in proof presentation. On the one hand, some formal systems are now seen as less important, and others as more important: for example, Gentzen systems are now widely used for presenting the rules about types in programming languages, and for similar purposes in the study of natural language, whereas Hilbert-style axiomatic systems are now not commonly used outside certain formal textbooks on logic for mathematicians. On the other hand, it is clear that even some natural deduction systems are inconvenient to implement on a computer — and when one looks in detail at why, one finds that the difficulties are often those which are hard to explain to students. For example, the widely used textbook by Lemmon presents a system of natural deduction rules, some of which are messy to implement and hard to explain.

MacLogic began as an attempt to understand the problems involved in implementing a proof assistant for type theory [as in the work of Martin-Löf], with a view to the correct development of programs from specifications. Such assistants exist (NuPRL, PICTT, lego, … - see Bibliography for details). It acquired an independent life of its own as a teaching package, and is intended (by its authors) as an introduction to the kind of proof construction that all computer scientists should do when learning logic, because essentially the same kind of work is involved in programming. But this "art of proof construction" is essentially that which anyone has to do in constructing proofs, whatever their discipline. (It is usually not the same as the way machines are programmed to do it automatically.)

We describe therefore in the remaining sections of this chapter what sorts of formal system MacLogic is really using, and how they are related.

4.2 Natural deduction calculi

Gentzen introduced in his pioneering papers in the 1930s the idea of a natural deduction proof. There are various notions: one version is that a proof consists of a tree of formulae, where the leaves are assumptions, and the discharge of an assumption (for a rule such as →I, for example) is indicated by striking it out. Those assumptions on which the conclusion (the formula at the tree's root) depends are just those not struck out.

This is convenient, in the sense of requiring little copying of formulae: but it is often not clear at a glance what assumptions any formula in the midst of the tree depends on. Gentzen therefore introduced a so-called "logistic" version of natural deduction, in which the *judgments* (the objects on which inference rules operate) are sequents of the form used in MacLogic: a formula list, then a special symbol such as ⊢, then a formula. Proofs are then defined as trees, where each node is such a sequent, and is either a leaf node, justified by the rule of Assumption, or is an internal node, justified from its children (the nodes just above it) by a rule of inference.

The notation we use in presenting proofs in the style of Lemmon is essentially a condensed version of this. First, the tree is squashed to a graph, to allow maximal sharing of sub-proofs: second, the graph is stretched out into a linear form, for ease of layout on the page, and third, the references back to assumptions are by line number rather than by using the formulae themselves.

The rules of natural deduction are generally of two kinds: rules for introducing a logical constant, and rules for eliminating one. The constants are introduced and eliminated only in the formula on the right of the turnstile ⊢. (In classical logic, there is an extra rule, **DN**, not fitting into this pattern, for removing two negation symbols at once. We avoid calling this rule an "elimination rule".)

See the monograph by Prawitz, or the survey by Sundholm, for further details of natural deduction.

4.3 Sequent calculi

The problem with natural deduction is that it is often not clear, even for simple problems, how to construct proofs. Ideally, you should be guided by your knowledge of the rules and the syntax of the problem to be solved. Gentzen therefore introduced a related system, called the *sequent calculus*, in which there are rules for introduction of the logical constants on the right of the turnstile, and (instead of elimination rules) rules for introduction of the constants on the left. (For details, see the literature, such as Gentzen's original papers, or the survey by Sundholm.)

To understand how MacLogic works, it is vital to recognise that the sequent calculus rules can be regarded, upside down, as tactics for decomposing problems: and that a sequent calculus proof can be regarded (at least in the intuitionistic case) as a recipe for constructing a natural deduction proof.

For example, the intuitionistic sequent calculus rule for "&-introduction on the left" is of the form

$$\frac{F,\ A,\ B,\ G\ \Rightarrow\ C}{F,\ A\&B,\ G\ \Rightarrow\ C}$$

where F, G are formula lists and A, B, C are formulae. Let us suppose that we can convert the proof leading to the premiss of this rule instance into a proof of C from the assumptions in F, from A, from B, and from those in G: we can now see how to construct a proof from the assumptions in F, from A&B and from those in G: we add A&B as an assumption, derive from it the formulae A, B by the natural deduction rule &E, and replace the uses (if any) of the assumptions A, B by appeals to A (resp. B) based on the new assumption of A&B. We therefore see the rule of "&introduction on the left" (in the sequent calculus) as an instruction to use the rule **&E** in constructing a natural deduction proof. Accordingly, the tactics in MacLogic are named, for example, "Tactic for &E", etc, rather than after the sequent calculus rules themselves. Below we shall often talk about use of a tactic for a natural deduction rule rather than naming the appropriate sequent calculus rule.

This view of sequent calculus rules as tactics for use of natural deduction rules works well for the intuitionistic case: we leave it to the interested reader to work out the correspondence for the other logical constants as an exercise. Accordingly, it is a simple mechanical task to convert a sequent calculus proof (as built in the **Derivation** window, with the root of the tree at the top left and the leaves at points marked by ■) into a natural deduction proof. This is done for you by MacLogic: but you should try to think, as you perform the simple proofs, just how the tactics you are using could be used to construct a natural deduction proof as you go along rather than at the end of the process. Note that the leaves of the derivation turn not into the assumption sequents in the natural deduction proof, but into the *minimal* sequents, in the middle of the proof, being (in general) both conclusions of elimination rules and premisses of introduction rules.

In the classical case, we use not Gentzen's sequent calculus LK, but that (LJ) for intuitionistic logic, augmented by a double negation rule (upside down, of course, as a tactic). The translation into a natural deduction proof is now easy — whereas if we used LK, with sequents having multiple consequents, the translation into natural deduction is non-trivial. (It is for this reason that MacLogic's automatic theorem prover gives no advice about how to solve problems: it is based, in the classical case, for efficiency, on the LK calculus, and although it could advise on how to solve a problem in the LK formalism, this is not particularly helpful.)

One very instructive example is the proof of the "Law of Excluded Middle", A∨~A, in classical logic. (In the sequent calculus LK, allowing multiple consequents on the right, this is trivial - the proof takes two lines: the first[7] step breaks up the disjunction, obtaining $\Rightarrow A,\sim A$; the next step breaks up the negation, obtaining $A \Rightarrow A$, which is an axiom.)

In the sequent calculus LJ augmented with a double negation rule DN, the first step (using the tactic for DN) is to pose the more complex problem $\Rightarrow \sim\sim(A\vee\sim A)$, now to use **~I**, obtaining the problem $\sim(A\vee\sim A) \Rightarrow \Lambda$, then **~E**, obtaining $\sim(A\vee\sim A) \Rightarrow A\vee\sim A$, then **∨I**, obtaining $\sim(A\vee\sim A)\Rightarrow\sim A$, then **~I**, obtaining $A, \sim(A\vee\sim A) \Rightarrow \Lambda$, then **~E**, obtaining $A, \sim(A\vee\sim A) \Rightarrow A\vee\sim A$, and (finally), **∨I** again. (Two trivial sub-problems have been omitted.)

As a sequent calculus proof (in LJ + DN) we have it as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A,\sim(A\vee\sim A)\Rightarrow A}}{A,\sim(A\vee\sim A)\Rightarrow A\vee\sim A}\quad\dfrac{}{\Lambda,A\Rightarrow\Lambda}}{A,\sim(A\vee\sim A)\Rightarrow\Lambda}}{\sim(A\vee\sim A)\Rightarrow\sim A}}{\sim(A\vee\sim A)\Rightarrow A\vee\sim A}\quad\dfrac{}{\Lambda\Rightarrow\Lambda}}{\sim(A\vee\sim A)\Rightarrow\Lambda}}{\Rightarrow\sim\sim(A\vee\sim A)}}{\Rightarrow(A\vee\sim A)}$$

and as a natural deduction proof, in Lemmon's notation, we have it as

1	(1)	~(A∨~A)	Ass
2	(2)	A	Ass
2	(3)	A∨~A	2 ∨I
1,2	(4)	Λ	1,3 ~E
1	(5)	~A	2,4 ~I
1	(6)	A∨~A	5 ∨I
1	(7)	Λ	1,6 ~E
	(8)	~~(A∨~A)	1,7 ~I
	(9)	A∨~A	8 DN

Note that there is a straightforward translation of the sequent calculus (in LJ + DN) proof into a natural deduction proof, where no similar translation is available from the proof in LK. Note also that, in the sequent calculus proof, as we move from the root to the leaves, we have on the left a formula $\sim(A\vee\sim A)$, which is broken up using the tactic for **~E**, giving us the formula $A\vee\sim A$ on the right (in the problem on the leftmost branch of the tree) and a trivial problem $\Lambda \Rightarrow \Lambda$ in the rightmost branch. But in the problem on the left, we have still the "broken up" formula $\sim(A\vee\sim A)$ in the antecedent, and this is broken up at the point further up the tree where the tree branches again.

This phenomenon, that as we break formulae up we cannot always throw them away, is an inconvenience of LJ (with or without DN): it adds significantly to the cost of automatic theorem proving in intuitionistic logic. It does not appear in LK (until we consider the effect of the quantifiers), which is why it is relatively easy mechanically to decide problems in zero-order classical logic. But it is also a reason why finding natural deduction proofs in classical logic can be non-trivial, even in the zero-order case: witness the above example.

---

[7] We are of course reasoning backwards from the goal, as in MacLogic's Construct mode - so "first" means "nearest to the root of the proof tree".

MacLogic's Construct mode allows you, in two ways, to control whether or not formulae are thrown away as you break them up. (Clearly, when a formula on the right of the ⇒ symbol is broken up, there is nowhere in the statement of the new sub-problem(s) to put the old formula.) When a formula on the left is the formula on which the tactic operates, it is removed if

- the **Autothin** check box in the dialog is checked, and
- the formula[8] is not an implication, a negation or a universal quantification.

Autothin's being checked is a sign to MacLogic that it can remove such formulae; but it will not do so if the formula is one of the kind to which the above comments about the possibility of their being needed again apply, *i.e.* is an implication, is a negation, or is universally quantified. If the formula is not removed, and you really want to clear it out of the way, then use the **Thin** tactic, via the **...** button on the dialog.

To repeat: MacLogic's Construct mode is based on the use of Gentzen's calculus LJ (perhaps with a DN rule for classical logic, or perhaps without the ∧-Intro_left rule—what we call the **∧E** tactic—for minimal logic), in which the rules are regarded upside down as tactics for building natural deduction proofs. (The idea is already in the work of Gentzen & Prawitz.)

4.4 Automatic theorem proving

MacLogic's theorem prover for classical logic is based on the sequent calculus LK of Gentzen. Like the rest of MacLogic, it is implemented in Prolog. The monograph by Melvin Fitting (see Bibliography) is an excellent coverage of this kind of approach. Difficulties arise with quantifiers, and we adopt the tricks involving Skolem functions and unification outlined by Fitting to reduce these difficulties. Nevertheless, MacLogic's theorem prover is not complete: but we believe that whatever answer it gives can be relied on, i.e. it can recognise when it has given an answer which may be wrong, and tells you so. Changing the level parameter may or may not give you a more informative answer.

Fitting's book actually outlines the methods of so-called "semantic tableaux": we regard these as a notational variant of the sequent calculus, with a trivial translation betwen the two.

For intuitionistic and minimal logic, roughly the same approach is adopted, using Gentzen's calculus LJ (or LM for minimal logic). In fact, a variant described elsewhere (see the paper by Dyckhoff mentioned in the Bibliography) is used instead, to avoid some problems with non-termination arising with the use of LJ.

The file "ATP Help" gives some more detail of how the automatic theorem prover can be controlled: access it on-line via the **ATP** item in the **Help** menu.

4.5 Bibliography

Burstall, R., …	*An interactive proof editor*, Computer Science Department, University of Edinburgh, 1986.
Constable, R., …	*Implementing Mathematics with the NuPrl Proof Development System*, Prentice-Hall, Inc., New Jersey 1986.
Dyckhoff, R.	*Contraction-free sequent calculi for intuitionistic logic*, Journal of Symbolic Logic **57** (1992), pp 795–807.

---

[8] We ignore here the presence of modal operators or equality.

Fitting, M.	*First-order logic and automatic theorem proving*, Springer-Verlag, 1990.
Gentzen, G.	*Collected papers*, ed. M. Szabo, North-Holland 1958.
Hamilton, A.	*PicTT - Programming in Constructive Type Theory*, Computer Science Dept, University of Stirling, 1989.
Hughes, G. E., …	*An introduction to modal logic*, Methuen, London, 1968.
Lemmon, E.	*Beginning Logic*, Nelson, London, 1965.
Lewis, C.I., …	*Symbolic Logic*, The Century Co., New York & London, 1932.
Martin-Löf, P.	*Intuitionistic type theory*, Bibliopolis, Naples, 1984.
Pollack, R.	*Lego - an implementation of the Calculus of Constructions*, Computer Science Department, University of Edinburgh, 1988.
Prawitz, D.	*Natural deduction*, Almquist & Wiksell, Uppsala, 1965.
Read, S.L. & Wright, C.	*FORMAL LOGIC*: an*introduction to first-order logic*, Logic & Metaphysics Dept, University of St Andrews, 1989 (& 1990, 1991, 1992, …)
Sundholm, G.	'Systems of Deduction', in *Handbook of Philosophical Logic*, vol I, ed. D. Gabbay & F. Guenthner; D. Reidel Pub. Co., Dordrecht, 1983, pp 133-188.
Zeman, J.J.	*Modal logic: the Lewis modal systems*, Clarendon Press, Oxford, 1973.

5. Reference Guide

5.1 Installation instructions

In this section, we explain how to install MacLogic on your Apple Macintosh and how to customise it for your (or your students') use. Some difficulties may arise, for example with memory management: if you experience these, and the information below fails to help you to sort out the problems, contact those responsible for maintaining the software by e-mail to the address at the end of this manual.

5.1.1 System requirements

5.1.1.1 Hardware and system software

MacLogic works under System Version 6.0.3, and some earlier versions. It also works under system 7, but without exploiting the special features thereof such as “Balloon Help”.

Ensure that your MacLogic disc is write-protected. Copy all the files from your MacLogic disc to either a hard or a floppy disc, and put your MacLogic disc in a safe place.

Under System 6, MacLogic runs on any Macintosh with at least 2 megabyte of RAM. (An older version runs in just 1Mby, but is no longer supported.)

Under System 7, MacLogic runs on any Macintosh with at least 4Mby of RAM.

You may need to ensure that your RAM cache is switched OFF. If this is not done, there may be memory allocation problems. To switch the RAM cache off, use the Control Panel from the (Apple) menu.

If there is not enough free RAM space in your Macintosh, MacLogic may either fail to load, or will load improperly and not be of any use. It may even load partially, but not have a File menu allowing you to quit gracefully: in this case, you have to reboot your Macintosh via the “reset switch”: for details of this procedure, see your Macintosh manual.

If MacLogic won’t load properly, run it on as large a machine as you can find and check (via the **About MacLogic…** item in the menu) that the evaluation space is set as low as possible (24K): change it to this if necessary, and transfer the copy of MacLogic back to the smaller machine. If there still are problems, contact us with details by e-mail.

MacLogic is MultiFinder-compatible (and System 7 compatible): a suggested size for the application memory is 1024Kbytes, but this can be increased if you can afford the space, or decreased a little if you must. Do this by selecting the MacLogic icon on the desk-top, and calling the **Get Info** command from the **File** menu: then edit the box at the bottom right.

5.1.1.2 Installing the logic fonts

The disc on which MacLogic is supplied contains a folder “Fonts for MacLogic”. This contains

- a suitcase file called “Detroit Fonts”, containing a 12 pt bit-mapped font **Detroit**, for the menus and dialogs. **Detroit** is a copy of **Chicago**, with extra characters for the logic symbols. Its font ID is 149.
- versions of the Konstanz font. Konstanz is a copy of Geneva, with the same extra characters as Detroit. Its font ID is 2500. It is available as 9 and 12 pt bit-maps in a suitcase, in TrueType format and in PostScript Type 1 format. At least install the bit-maps; install the others if you want to print MacLogic documents (or this documentation).

For best results, these two fonts should be installed in your System file. How to do this varies between System 6 and System 7: see your Macintosh manual for details. BEGIN BY DISCARDING **ALL** OLD VERSIONS OF KONSTANZ AND DETROIT[9] IN YOUR SYSTEM FILE. (If you must, keep copies in a safe place.) Older versions of MacLogic had the fonts attached to the application

---

[9] Copies, if any, of Detroit (New) should be removed at the same time.

MacLogic itself: this still works fairly well, except for some minor problems with MultiFinder. The recommended method now is to install the fonts in your System file.

Once the fonts are installed, you can of course use them while writing your logic essays with a word processor.

5.1.1.3 Memory management

When MacLogic has been loaded, you can change the amount of "evaluation space" used. Memory used by MacLogic is of two kinds - "evaluation space" and "free memory". You can increase the former, thereby decreasing the latter, and *vice versa*: it is best to have them about equal. The changes can be made from a button on the initial dialog, or from the **About MacLogic…** item in the menu. Once it has been changed, the new setting is saved (on quitting MacLogic, but only if MacLogic is not a read-only file) in the MacLogic application file as a resource item for use next time. So, when installing MacLogic on a machine with lots of memory, you may wish to make this change immediately, quit from **MacLogic**, and then lock the file to prevent it from being changed (other than temporarily) again. The minimum setting you can use is 24K, but for serious work you should aim for 64K or more. I use 128K.

Warning: if you increase the setting, and then move that copy of MacLogic to a different machine, with less memory (or even just increase the size of the System file), there may be problems. Your original copy of MacLogic should still be usable, however.

5.1.2 What files are needed, where

MacLogic is distributed as a demonstration version. Details of how to upgrade it to a full working version are available to licence holders.

The code file necessary for MacLogic to work is

• MacLogic 2.2	the application itself

There are some help files, without which the on-line help system will not be usable. These live in a folder called HELP, and are called:

• ATP Help	about the Automatic Theorem Provers
• Menu Help	about all the menu items
• Rules Help	about the inference rules
• Syntax Help	about the syntax
• Tactics Help	about the tactics

See section 5.1.3.2 below for details of how these help files can be edited.

The code file should be kept in a folder, which should also contain the HELP folder. Do not drag MacLogic onto the desktop and then try to run it: when it comes to looking for the HELP folder, it expects to find it in the same folder as itself. **Leave it in its own folder! (**You may find that an application launching utility such as *On Cue*™ is a useful alternative to having MacLogic on the desktop. Under system 7, you can of course create an *alias* and put that on the desktop, or in the Apple Menu folder.)

There is also a folder of fonts:

• Fonts for MacLogic	contains Detroit and Konstanz fonts

As noted above, some of the contents of this folder should be installed in your system file before you use MacLogic.

The following additional files may be available:

• MacLogic Problems	a problem library
• MacLogic Settings	menu settings
• MacLogic Theorems	consisting of saved theorems

but, if not available, they can be generated quite easily when you run **MacLogic**.

5.1.3 Customisation instructions

MacLogic can be customised for your own (or your students') use in various ways. You can create a file "MacLogic Settings" which records the state of all the options in the **Logic** menu and the **Options** menu, for example if you always prefer to use Construct mode and Classical Logic, and to have your windows in 9pt. You can also edit the help files, if you don't like the way in which the rules *etc* are explained. You can build libraries of problems of your own. You can save theorems for use with the rule "Sequent Introduction". How to do all this is described below.

5.1.3.1 MacLogic Settings

When MacLogic is being loaded, it looks for a file "MacLogic Settings" in its home folder. The file should contain some code (in Prolog) recording how the various menu options were set when it was last modified.

If the file is present, and is of the right kind, the contents are interpreted and the menu settings of MacLogic are reset accordingly.

When you exit from MacLogic, and have changed any of the menu settings, you are asked if you want to save the settings. If you answer **Ok**, then you can save the settings to a file of whatever name you like, in any folder or disc to which you have appropriate access. You will need to rename this file to "MacLogic Settings" at a later stage, and move it back to MacLogic's home folder (Note that the correct name is NOT "MacLogic settings" or "Maclogic Settings".)

If using System 7, do not put the settings file into the Preferences folder and expect MacLogic to find it.

(It is not necessary to know anything about Prolog to be able to create or use this file.)

5.1.3.2 Changing the Help information

The HELP folder contains a number of help files, which you are free to edit. Note the format carefully: only edit the bits in between the comment marks /* and */. Save them as TEXT documents if you use a word processor to do the editing. The best way is to use MacLogic itself as a text editor, and then the files are saved as TEXT documents automatically. Ensure that your new versions have the correct names (as in section 5.1.2 above), with the correct upper case/lower case distinctions. When saving from MacLogic you will have to use new names like “New Tactics Help” for a while.

5.1.3.3 Building and editing problem libraries

MacLogic comes with a standard library of problems, called “MacLogic Problems”. This is pre-loaded automatically, if it is kept in the home folder, when MacLogic is loaded. You can create similar problem libraries for yourself.

Library files are of two kinds: textual and coded. The textual form is in case you want to edit or print it with a word-processor. The coded form is for speed of use by MacLogic. MacLogic can convert between the two forms for you.

The format of the textual form is a sequence of lines containing problems. A *problem* is

- either a formula, or
- a sequence of formulae (separated by commas), then either a turnstile ⊢ or a sequent arrow ⇒, and then a formula.

Any part of a line including and following a % symbol is ignored, so you can include comments of your own. Blank lines are ignored. Note that a problem must fit on a single line, as indicated by the Carriage Return character rather than the visual appearance. If a problem is being read for storage in the library, there is a restriction of its being at most 255 characters long. (Larger problems can be tackled by MacLogic, so long as the formulae don’t exceed 255 characters, but they can’t be put in the library.)

To create a library file from scratch, use the **Create…** command from the **Windows** menu of MacLogic, and choose a name such as “My Library” for the new window. Type the problems into the window in the format just described. You can check as you go along that each problem is syntactically correct, and even whether or not it is solvable, by using the standard features of MacLogic, such as **Valid…** from the **Help** menu. Save the file, using the **Save to text file…** item in the **File** menu.

Leave the window as the front window. To convert it to a coded form, use the command **Load library problems…** from the **File** menu, and select the option of loading from the front window. It will be read by MacLogic, and the information stored in memory, for immediate use. This takes some time, since the problems you have typed have to be parsed (i.e. converted from textual to coded form). Now use **Save library problems…** from the **File** menu, and save to a new coded file, such as “My Library Coded”. You may wish to rename this later to “MacLogic Problems”, having first put the old file of that name in a safe place.

To edit an existing library file, the method depends on whether it is textual or coded. If it is textual, open it as a text file (**Open text file…** from the **File** menu), then edit it, and save it as a text file. (The steps just described above can be used to parse it into a coded form suitable for future use.) If it is coded, load it into library (**Load library problems…** from the **File** menu), save it as text file, and proceed as above.

Library files can of course be printed. Text files can be loaded, as just described, and then printed (**Print visible windows…** from the **File** menu.) Coded files can be loaded into memory (**Load library problems…** from the **File** menu), then saved to a text file (**Save library problems…** from the **File** menu), then loaded into a window and printed as just described.

The folder “Problem files” contains several examples of libraries, all in textual form. Some are routine problems: some are rather hard. Some are unsolvable, being used for testing the automatic theorem proving facilities.

5.1.3.4 Building theorem files

When you use MacLogic, it is often convenient to appeal to a theorem proved earlier by solving a problem. So, as you solve problems, you may wish to keep them as theorems for future use. This is done by ensuring that the item **Keeping as theorem** in the **Options** menu is ticked. All zero-order[10] problems that you solve will then be remembered, provided you give them names when prompted to do so.

On quitting MacLogic, you are asked if you want to save all your theorems. If you answer **Ok**, an appropriate file is created. You may wish to rename this to “MacLogic Theorems”, having first kept a safe copy of the previous version of the file..

On loading MacLogic, if a file “MacLogic Theorems”, of the right type and with the right contents, is in the folder to which MacLogic belongs, then it is loaded as a database of theorems.

If you wish to prove lots of theorems rather fast, create a library window containing them all as problems, ensure that **Keeping as theorem** is ticked on the **Options** menu, and use the automatic mode (using **Help/Valid**) to solve all the problems one by one, naming them as you go along. Better still: you’ll find an item in the **Problem** menu called **Test run**: if you select this, it attempts to solve all the problems in the library, reporting in the **Jotter** window the time taken for each problem. However, if your theorems are in different logics, this will be inconvenient, and you’ll have to get them proved one by one, adjusting the logic each time to the most primitive logic in which the result is provable.

---

[10] Remember, this means "of propositional calculus", i.e. without individual variables or quantifiers. There is no reason in principle why this restriction is imposed, but it is convenient: a later edition of MacLogic will omit the restriction.

5.1.3.5 Foreign language environments

There is no convenient way of customising MacLogic to work in a language other than English. You can edit the help files, but the dialogs are not editable, being not stored as resources but as Prolog code.

If your Macintosh is organised for languages other than English, it may be necessary to work out what keystrokes are needed for the different logical symbols. This should be obvious from **KeyCaps** or (better) a utility like **KeyFinder** or **PopChar** and information about the Syntax is available from the **Help/Syntax** menu item or in the file "Syntax Help"—but the assignment therein of keystrokes to special symbols will have to be worked out for your keyboard. In case all else fails, here are the ASCII codes for the interesting symbols:

Absurdity	Λ	236	(*f*	[196]	also allowed)
Negation	~	241	(¬	[194]	also allowed)
Disjunction	∨	235			
Conjunction	&	38			
Implication	→	231	(⊃	[250]	also allowed)
Equivalence	↔	234			
Equality	=	61			
Universal Quantifier	∀	232			
Existential Quantifier	∃	228			
Necessity	□	246			
Possibility	◊	215			
Strict implication	⥽	225			
Turnstile	⊢	230			

MacLogic works best with a British keyboard.

5.2 Syntax

In this section we give the contents of the “Syntax Help” file, which can be seen on-line if you wish.

MacLogic uses a special font called 'Konstanz', which has, *inter alia*, the following LOGICAL CONSTANTS:-

→	“Shift Option Y”	for implication
&	“Shift 7”	for conjunction
∨	“Shift Option D”	for Disjunction
~	”Shift Option L”	for negation
↔	”Shift Option S”	for biconditional
∃	“Shift Option E”	for the Existential quantifier
∀	“Shift Option U”	for the Universal quantifier
Λ	“Shift Option F”	for absurdity (Falsehood)
□	“Shift Option N”	for Necessity
◊	“Shift Option V”	for possibility
⥽	“Shift Option 9”	for strict implication
=	“=”	for equality[11]
⊢	“Shift Option T”	for Turnstile
⇒	“Shift Option 2”	for Sequent Arrow

MacLogic will also allow you to type

⊃	“Option H”	for implication,
¬	“Option L”	for negation, and
f	“Option F”	for absurdity,

but these will be displayed in proofs as '→', '~' and 'Λ' respectively.

ALPHABETIC characters are interpreted as follows:

a, ..., z	are INDIVIDUAL letters, standing for “individuals”.
A, ..., Z	are PREDICATE or PROPOSITION letters, standing for “predicates”.

Individual letters will also be called VARIABLES. We use the Greek letter μ schematically below to indicate a variable. There is no special category of “constants”.

---

[11] Elsewhere often called “identity”).

A TERM consists just of an individual letter. (More powerful logics, but not MacLogic, allow compound terms.)

An ATOMIC FORMULA consists of a predicate or proposition letter followed by a finite (possibly, and in propositional logic necessarily, zero) number of terms.

[WELL-FORMED] FORMULAE (i.e. WFFs) are inductively defined as follows:-

(a) any atomic formula is a wff;
(b) Λ is a wff;
(c) if A is a wff, then $\sim A$ is a wff;
(d) if A and B are wffs, then $(A\rightarrow B)$ is a wff;
(e) if A and B are wffs, then $(A\&B)$ is a wff;
(f) if A and B are wffs, then $(A\vee B)$ is a wff;
(g) if A and B are wffs, then $(A\leftrightarrow B)$ is a wff;

(h) (for Predicate Logic only) let A be a wff containing the variable μ, but not containing a quantification of the form $(\forall\mu)$ or $(\exists\mu)$; then $(\forall\mu)A$ and $(\exists\mu)A$ are wffs. The wff A is called the SCOPE of the quantification. We say that a quantification, $(\exists\mu)$, or $(\forall\mu)$, BINDs every occurrence of μ within its scope. These occurrences of variables are called BOUND. (Occurrences of) variables which are not bound are FREE. A term t is said to be FREE FOR a variable μ in A if no occurrence of μ in A lies in the scope of a quantification of the form $(\forall t)$ or $(\exists t)$. Loosely, one also talks of the SCOPE of a quantifier.

Note that rule (h) forbids:

repeated quantification, as in $(\forall x)(\forall x)Fxx$,
vacuous quantification, as in $(\forall x)(\exists y)Fx$, where 'y' is vacuously quantified.

(if Equality is in use)

(i) if s and t are terms, $(s = t)$ is a wff;

(if Modal logic is in use)

(k) if A is a wff, then $\Box A$ is a wff;
(l) if A is a wff, then $\Diamond A$ is a wff;
(m) if A and B are wffs, then (A ⥽ B) is a wff.

Definitions of MODAL and COMODAL :

(1) (i) □A, ~◊A, (A ⥽ B) and Λ are S4-modal;

(ii) if A and B are S4-modal, so are

A&B, A∨B and (∃μ)A;

(2) (i) □A, ◊A, A ⥽ B and Λ are S5-modal;

(ii) if A and B are S5-modal, so are

~A, (A&B), (A∨B), (A→B), (∀μ)A and (∃μ)A.

(3) (i) ◊A, ~□A, ~(A⥽B) and ~Λ are S4-comodal;

(ii) if A and B are S4-comodal, so are (A&B), (A∨B) and (∀μ)A;

(4) Every S5-modal wff is S5-comodal.

If one is using the modal logic S4, then 'appropriately modal' means S4-modal; similarly in S5.

Parentheses may be omitted in certain cases: one may omit outermost parentheses, and may drop inner parentheses where possible by virtue of the ranking of propositional connectives: '~' ties more closely than '&' or '∨', which tie more closely than '→', which ties more closely than '↔'. Thus 'P&~Q→R' abbreviates ((P&~Q)→R). The quantifiers rank so that (∀x)Ax&B abbreviates ((∀x)Ax)&B, rather than (∀x)(Ax&B). If in doubt, put the parentheses in.

Also, '→' is right associative, which means that A→B→C means the same as A→(B→C); but '&' and '∨' are left associative, so that A&B&C means the same as (A&B)&C (and similarly for '∨'). '⥽' is right associative too. When a formula is displayed by MacLogic, some extra space is put round the main connective to help you see the structure, as in A&B & C.

Square braces '[', ']' may be used sensibly instead of parentheses, as in (∀x)[Ax&(Bx→Cx)]: this helps make the formula’s structure more clear. They must occur in matching pairs, and may not be used around quantifiers, as in [∀x]Ax.

5.3 Rules of first-order logic, as implemented

Rules in General:

A proof consists of a succession of consecutively numbered lines, each representing a sequent.

A "sequent" consists of a formula asserted on the basis of certain assumptions (which are, in turn, formulae) and justified by appeal to a rule of inference (and perhaps the citation of some premisses, i.e. sequents arising earlier in the proof). The assumptions are referred to by number; so also are the premisses. For example, the line (number 4 of the proof)

1	(4)	A&B	3,4 &I

appearing in a proof represents the sequent "the formula A&B is asserted on the basis of the assumption of the formula on line 1, and follows from the sequents on lines 3 and 4 by the rule &I".

Rule of Assumption:

This rule permits us to introduce at any stage of an argument any proposition we choose as an assumption. No premisses are needed:

1	(1)	P	Ass
	...		
6	(6)	~Q	Ass

P and ~Q here each depends on itself as assumption.

Rule of &I:

Given proofs of any two formulae A, B, we can append their conjunction A&B as a conclusion. The assumptions in the two premisses are pooled to give the assumptions of the conclusion. Note that the numbers of the premiss lines must be cited in the same order as the formulae themselves appear in the conjunction:

1	(1)	P	Ass
2	(2)	Q	Ass
1,2	(3)	Q&P	2,1 &I

Rule of &E:

Given a proof of any conjunctive formula A & B as a premiss, we can prove conjunct A (and also conjunct B) as a conclusion, depending on the same assumptions as the premiss:

1	(1)	P&Q	Ass
1	(2)	Q	1 &E
1	(3)	P	1 &E

Rule of $\vee$I:

Given a proof of any formula as a premiss, we can prove a disjunction consisting of that formula disjoined with any formula as a conclusion:

1	(1)	P	Ass
1	(2)	$Q\vee P$	1 $\vee$I
1	(3)	$P\vee Q$	1 $\vee$I
1	(4)	$P\vee P$	1 $\vee$I

Rule of $\vee$E:

Let A, B and C be any three formulae, and suppose

a) that we have a proof of $A\vee B$,

b) that on assuming A we can prove C as a conclusion,

c) that on assuming B we can prove C as a conclusion;

then we can draw C as a conclusion from any assumptions on which $A\vee B$ rests, together with:

i) any assumptions (apart from A itself) on which C rests in its proof from A, and

ii) any assumptions (apart from B itself) on which C rests in its proof from B.

	…		
1,4	(6)	$P\vee Q$	5 &E
7	(7)	P	Ass
3,7	(8)	R	3,7 $\rightarrow$E
9	(9)	Q	Ass
2,9	(10)	R	2, 9 $\rightarrow$E
1,2,3,4	(11)	R	6,7,8,9,10 $\vee$E

Note the strict ordering of the justification lines, the premisses, cited at line 11 - the line containing the disjunction (6), the assumption of the first disjunct (7), the conclusion drawn from the first disjunct (8), the assumption of the second disjunct (9), and finally the conclusion drawn from the second disjunct (10).

Rule of $\rightarrow$I:

If some formula B depends on a set X of assumptions, then we may remove the assumption A from X and obtain the conclusion $A\rightarrow B$ on the remaining assumptions (if any). The assumption A is said to be *discharged*. (If A is not in X then removing A from X leaves X unchanged.)

1	(1)	P	Ass
2	(2)	Q	Ass
1,2	(3)	$P\&Q$	1,2 &I
1	(4)	$Q\rightarrow(P\&Q)$	2,3 $\rightarrow$I

The first premiss cited must be that where the discharged assumption, the antecedent of the conditional, was assumed, and the second premiss that where the consequent (of the conditional) was proved.

Rule of →E:

Given proofs of a conditional formula A → B, and of the antecedent A of that conditional, we may obtain the consequent B of the conditional as a conclusion. The assumptions are pooled:

6	(6)	P→Q	Ass
2	(7)	P	2 &E
2,6	(8)	Q	6,7 →E

The first premiss cited must be the line containing the conditional itself, and the second that containing its antecedent.

Rule of ~I:

If from some assumptions X, we can obtain Λ as a conclusion, then we can remove the assumption A from X and obtain ~A as a conclusion from the remaining assumptions, if any. (If A is not in X, then removing A from X leaves X unchanged.) The assumption A is said to be *discharged*.

1,2	(6)	Q	1,2 →E
3	(7)	~Q	3 &E
1,2,3	(8)	Λ	7,6 ~E
1,2	(9)	~(P & ~Q)	3,8 ~I

The first premiss cited must be that where the discharged assumption, the formula whose negation is proved, was assumed, and the second premiss the line where Λ was proved.

Rule of ~E:

Given a proof of A, and a proof of its negation ~A, we can draw Λ as a conclusion. The assumptions are pooled, and the lines at which ~A, A occur are cited (in that order) as the premisses.

1,3	(6)	~P	4 &E
2,5	(7)	P	2,6 →E
1,2,3,5	(8)	Λ	6,7 ~E

Rule of ΛE:

Given a proof of absurdity, one may obtain any formula whatever as a conclusion. The conclusion rests on the same assumptions as the premiss.

1,3	(5)	Λ	...
1,3	(6)	P	5 ΛE

ΛE is not available in minimal logic. In classical logic it is not regarded as a primitive rule, but as a rule derivable from the rule DN. The above piece of proof in classical logic would then be:

1,3	(5)	Λ	...

6	(6)	~P	Ass.
1,3	(7)	~~P	5,6 ~I
1,3	(8)	P	7 DN

Rule of DN:

Given a proof of the double negation of a formula, we can obtain the formula itself as a conclusion. Only one premiss is needed for this rule. The conclusion rests on the same assumptions as the premiss.

1	(1)	~~A	Ass
1	(2)	A	1 DN

DN is not available in either intuitionistic or minimal logic.

Rule of ∀I:

If an arbitrarily selected object, for example 'y', can be shown to have a property, A(y), then everything must have it, that is, (∀x)A(x), where 'x', 'y' are any variables. Before we apply ∀I in passing from a proposition about 'y' to a universal conclusion, we must make sure that the 'y' does not occur free in any of the assumptions in the premiss:

1	(1)	(∀x)(Fx→Gx)	Ass
2	(2)	(∀x)(Gx→Hx)	Ass
1	(3)	Fx→Gx	1 ∀E
2	(4)	Gx→Hx	2 ∀E
1,2	(5)	Fx→Hx	3, 4 SI Hyp Syll
1,2	(6)	(∀x)(Fx→Hx)	5 ∀I

Rule of ∀E:

If everything has a certain property, any particular thing must have it, and so we can pass, for example, from a proof of (∀x)A(x) to the conclusion A(t), where x is a variable and t is any term, provided t is free for x in A(x), that is, provided t is not a variable such that x occurs in A(x) within the scope of a quantifier over t. [If "Alpha-conversion" is enabled, then any occurrences of t inside A(x) can be renamed at the same time, thus allowing the substitution of t for x without breaking this proviso.]

Only one line is cited as a premiss, that where the formula (∀x)A(x) was proved. The assumptions in the conclusion are the same as those in the premiss.

1,2	(3)	(∀x)(∃y)(Hx&Gy)	1,2 →E
1,2	(4)	(∃y)(Hx&Gy)	1 ∀E
1,2	(5)	(∃y)(Hz&Gy)	1 ∀E
1,2	(6)	(∃y)(Ha&Gy)	1 ∀E

Note that we may NOT conclude (∃y)(Hy&Gy), for y is not free for x in (∃y)(Hx&Gy). [If alpha-conversion is enabled, then we may conclude (∃z)(Hy&Gz).]

Rule of ∃I:

If a particular thing has a certain property, then something must have it. Thus ∃I permits us to pass from a derivation of the premiss $A(t)$ to the conclusion $(\exists x)A(x)$, where t is any term and x is any variable.

1	(1)	(∀x)Hx	Ass
1	(2)	Ha	1 ∀E
1	(3)	(∃x)Hx	2 ∃I

(Similar reservations about t being free for x in $A(x)$ apply, as in the case of ∀E.)

Rule of ∃E:

If something has a certain property, and if it can be shown that a conclusion C follows from the assumption that an arbitrarily selected object has that property, then we know that C holds. Thus, given, for example, $(\exists x)A(x)$, we assume a typical instance $A(y)$, and if we can prove C from $A(y)$, then C will follow from $(\exists x)A(x)$. The conclusion C will rest on any assumptions on which $(\exists x)A(x)$ rests, and on any assumptions used to derive C from the corresponding typical instance $A(y)$, apart from the instance $A(y)$ itself.

On the right-hand side, we cite three lines:

- i) the line where the existential formula $(\exists x)A(x)$ occurs;
- ii) the line where the typical instance $A(y)$ is assumed; and
- iii) the line where C is drawn as a conclusion from the typical instance $A(y)$ as assumption.

Note that y must be chosen so that it is free for x in $A(x)$ [but if alpha-conversion is enabled, you can rename bound occurrences of y in $A(x)$ to avoid problems]. Note also that the variable y used must not occur free in the conclusion C drawn, nor in the assumptions used to derive C from the typical instance $A(y)$ (although of course it will appear in the assumption $A(y)$ itself), nor in $(\exists x)A(x)$, although it may appear bound in any of these places.

1	(1)	(∀x)((Fx&Hx)→Gx)	Ass
2	(2)	(∃x)(Fx&Hx)	Ass
3	(3)	Fx&Hx	Ass
1	(4)	(Fx&Hx) → Gx	1 ∀E
1,3	(5)	Gx	4,3→E
1,3	(6)	(∃x)Gx	5 ∃I
1,2	(7)	(∃x)Gx	2,3,6 ∃E

Rule of =I:

This rule is only available if 'Equality' is ticked in the 'Logic' menu. To use '=I' in Check mode select the '=…' button in the Next Line dialog box and respond to the ensuing dialog.

We may at any time add the fact that `t = t`, for any term `t`, to a proof. It rests on no assumptions:

(8) z = z =I

Rule of =E:

This rule is only available if 'Equality' is ticked in the 'Logic' menu. To use '=E' in Check mode select the '=…' button in the “Next Line” dialog box and respond to the ensuing dialog.

If we have a proof of s = t and another of A(s) (i.e. A(v) with s in place of some free variable v), then we may conclude A(t). The assumptions are pooled:

1	(1)	x = y	Ass
2	(2)	(∀z)Fxz	Ass
1,2	(3)	(∀z)Fyz	1,2 =E

A(v) is here (∀z)Fvz; so A(x) is (∀z)Fxz and A(y) is (∀z)Fyz.

Rule of =symm:

This rule is only available if 'Equality' is ticked in the 'Logic' menu. To use '=symm' in Check mode select the '=…' button in the Next Line dialog box and respond to the ensuing dialog.

Equality is symmetric so if we have a proof of s = t, where s and t are any terms, then we may conclude t = s, on the same assumptions :

1	(1)	a = y	Ass
2	(2)	y = a	1 =symm

Rule of =trans:

This rule is only available if 'Equality' is ticked in the 'Logic' menu. To use '=trans' in Check mode select the '=…' button in the “Next Line” dialog box and respond to the ensuing dialog.

Equality is transitive so if we have a proof of s = t and another of t = u, where s, t and u are any terms, then we may conclude s = u. The assumptions are pooled:

1	(1)	a = y	Ass
2	(2)	y = z	Ass
1,2	(3)	a = z	1,2 =trans

Rule of Sequent Introduction:

This is not itself an extra rule, so much as a metalogical principle for creating further rules. It permits us to introduce, at any stage of a proof, a substitution instance of any propositional sequent recorded in a Theorems file (or proved earlier in a session), if that instance’s antecedents have been obtained earlier in the proof. So, if we have obtained in a proof A1, A2, .., An on various assumptions, and supposing that A1, A2, .., An ⊦ B is a (substitution-instance of a) sequent in the Theorems file, or for which we already have a proof, then we can draw B as a conclusion on the pool of assumptions on which A1, A2, .., An rest. We cite on the right the lines on which A1, A2, .., An are proved (in that order), then 'SI' followed by the name of the sequent proved. Note that we only allow the version of SI where the sequent belongs to propositional logic, but that the substitution instance may belong to full predicate logic.

1	(1)	(∀z)Raz	Ass
	(2)	Fx∨~Fx	SI ExMid
1	(3)	(∀z)Raz & (Fx∨~Fx)	1,2 &I
1	(4)	Fx→(∀z)Raz	1 SI PMI

To use 'Sequent Introduction' in Check mode select the '…' button in the “Next Line” dialog box and respond to the ensuing dialog.

Rule of Tautology:

If one has proved A1, …, An , and A1, …, An ⊦ C is a valid sequent of the current logic, then one may conclude C on the basis of the assumptions on which A1, …, An rest.

To use “Tautology” in Check mode, select the ‘…’ button in the Next Line dialog box and respond to the ensuing dialog.

Rule of Substitution:

If one has proved a formula A, and has also proved a formula C (maybe on the assumption of A), then one may use the “Substitution” rule to substitute the first proof in place of the assumption (if any) of A, thus obtaining a proof of C, dependent on those assumptions on which A depends, together with those on which C depends (apart from A itself).

In sequent notation:

Gamma ⊦ A   A, Delta ⊦ C
------------------------Subs.
Gamma, Delta ⊦ C

Example:

1	(1)	A & B	Ass.
1	(2)	A	1 &E
3	(3)	A	Ass.
3	(4)	A & A	3,3 &I
1	(5)	A & A	2,3,4 Subs.

The numbers cited on the right (here, in line 5) refer to the line (here, 2) where “A” is proved, the line (here, 3) where it is assumed and the line (here, 4) where the other formula “C” is proved.

This rule is especially useful in combination with rules for modal logics. It is a special case of the S.I. rule; that, in turn can be obtained by several application of this rule.

To use this rule in Check mode, select the ‘…’ button in the “Next Line” dialog box and respond to the ensuing dialog.

Rule of Definition:

The expression A↔B is used as an abbreviation of (A→B) & (B→A). We may expand or contract all or part of a formula using this definition, and cite the rule “Defn.” in justification. Similarly, ~A is a definitional abbreviation of A → Λ.

1	(1)	P	Ass
2	(2)	(P∨R)&(P↔Q)	Ass
2	(3)	(P∨R)&((P→Q)&(Q→P))	2 Defn. ↔
2	(4)	(P→Q)&(Q→P)	3 &E
2	(5)	P→Q	4 &E
1,2	(6)	Q	5,1 →E

or, in the other direction:

1	(1)	P→Q	Ass
2	(2)	Q→P	Ass
1,2	(3)	(P→Q)&(Q→P)	1,2 &I
1,2	(4)	P↔Q	3 Defn ↔

To use “Definition” in Check mode, select the ‘…’ button in the “Next Line” dialog box and respond to the ensuing dialog.

The rule is applicable if the formula being derived is definitionally equal to that in the single cited premiss, and the assumptions are those of the premiss. Two formulae are definitionally equal if one can be obtained from the other by a sequence of expansions and contractions of definitional abbreviations.

If “Delta conversion” is enabled, much use of this rule is unnecessary.

Rule of □I:

This rule is only available in the modal logics S4 and S5. To use '□I' in Check mode select the '…' button in the “Next Line” dialog box and respond to the ensuing dialog.

If some proposition A depends on a set X of assumptions, where all the assumptions in X are appropriately modal, then we may conclude □A, from the same assumptions:

X ⊢ A

----- (all formulae in X are appropriately modal)

X ⊢ □A

1	(1)	□P	Ass
2	(2)	P⥽R	Ass
1	(3)	P	2 □E
1,2	(4)	R	2,3 ⥽E
1,2	(4)	□R	4 □I

(Note: the rule as implemented in Check mode is more general than the rule stated here. It says that on every path P from A to a member of X there is an appropriately modal formula C depending on no assumptions not in X and containing no free occurrence of a variable that is the eigenvariable of an inference on the path P. For details, see Prawitz' book "Natural Deduction", page 79.)

Rule of □E:

This rule is only available in the modal logics S4 and S5. To use '□E' in Check mode select the '…' button in the "Next Line" dialog box and respond to the ensuing dialog.

Given as a premiss □B, we may conclude B, from the same assumptions:

6	(6)	□Q	Ass
6	(7)	Q	6 □E

Rule of ◊I:

This rule is only available in the modal logics S4 and S5. To use '◊I' in Check mode select the '…' button in the "Next Line" dialog box and respond to the ensuing dialog.

Given as a premiss B, we may conclude ◊B, from the same assumptions:

1	(1)	P	Ass
2	(2)	P→R	Ass
1,2	(3)	R	2,1 →E
1,2	(4)	◊R	3 ◊I

Rule of ◊E:

[This rule is only available in the modal logics S4 and S5.] To use '◊E' in Check mode select the '…' button in the "Next Line" dialog box and respond to the ensuing dialog.

If (on the assumption set X) A is possible (i.e. ◊A is true) and from the assumption A (and some other assumption set Y) we can prove the conclusion B, then we may discharge the assumption A and prove the conclusion B on the assumption set X∪Y;

provided that

i) the conclusion B is appropriately co-modal and

ii) all the assumptions in Y are appropriately modal.

In sequent notation:

X ⊦ ◊A   A, Y ⊦ B

X, Y ⊦ B

Example:

1	(1)	◊(P & ~ P)	Ass
2	(2)	P & ~ P	Ass
2	(3)	P	2 &E
2	(4)	~P	2 &E
2	(5)	Λ	4,3 ~E
1	(6)	Λ	1,2,5 ◊E

(Note: the rule as implemented in Check mode is in fact more general than the rule stated here, in a manner similar to that described for **□I**. Here is an example:)

1	(1)	□(P&R→Q)	Ass
2	(2)	A	Ass
3	(3)	A→◊P	Ass
4	(4)	C	Ass
5	(5)	C→□R	Ass
2,3	(6)	◊P	3,2 →E
7	(7)	P	Ass
1	(8)	P&R→Q	1 □E
4,5	(9)	□R	5,4 →E
4,5	(10)	R	9 □E
4,5,7	(11)	P&R	7,10 &I
1,4,5,7	(12)	Q	8,11 →E
1,4,5,7	(13)	◊Q	12 ◊I
1,2,3,4,5	(14)	◊Q	6,7,13 ◊E

Rule of ⥽I:

This rule is only available in the modal logics S4 and S5. To use '⥽I' in Check mode select the '…' button in the “Next Line” dialog box and respond to the ensuing dialog.

If some formula B depends on a set Y of assumptions, where all the assumptions in Y, other than A, are appropriately modal, then we may discharge the assumption A from Y and obtain A⥽B as conclusion on the remaining assumptions (if any).

1	(1)	□Q&□R	Ass
2	(2)	P	Ass
1	(3)	□R	1 &E
1	(4)	R	3 □E
1	(4)	P⥽ R	2,4 ⥽I

The first premiss (here, 2) cited must be the line where the antecedent A is assumed, and the second (here, 4) that where its consequent B is obtained.

See the help about □I for details of how the rule actually implemented in Check mode is more liberal.

Rule of ⥽E:

This rule is only available in the modal logics S4 and S5. To use '⥽E' in Check mode select the '…' button in the "Next Line" dialog box and respond to the ensuing dialog.

Given a proof of a strict implication A ⥽ B, and of the antecedent A of that strict implication, we may obtain the consequent B of the conditional as a conclusion. The assumptions are pooled:

6	(6)	P⥽Q	Ass
2	(7)	P	2 &E
2,6	(8)	Q	6,7 ⥽E

The first premiss cited must be that where the strict implication itself was obtained, and the second that where its antecedent was obtained.

5.4 Tactics for first-order logic, as implemented

Tactics in General

Tactics are used in Construct mode for breaking a problem into sub-problems.

A problem is of the form $F \Rightarrow C$, where F is a list of formulae, the *facts*, and C is a formula, the *goal*. We write $B :: F$ to indicate the list with head B, tail F. F is sometimes called the *fact-list* of the problem.

Tactic for &I

The tactic for '&I' : replace a problem $F \Rightarrow A\&B$ by the two sub-problems $F \Rightarrow A$ and $F \Rightarrow B$.

In other words, to prove a conjunction $A \& B$, try to show A and B separately.

Tactic for &E

The tactic for '&E' : if a formula $A\&B$ is in F, then the problem $F \Rightarrow C$ is replaced by the problem $A :: B :: F \Rightarrow C$. (And, maybe $A \& B$ is removed from the list F).

In other words, if you already know $A \& B$, then add A and B to the facts you know.

Tactic for ∨I_left

The '∨I_left' tactic: replace the problem $F \Rightarrow A \vee B$ by the problem $F \Rightarrow A$.

In other words, to show $A \vee B$, try to show A.

Tactic for ∨I_right

The '∨I_right' tactic: replace the problem $F \Rightarrow A \vee B$ by the problem $F \Rightarrow B$.

In other words, to show $A \vee B$, try to show B.

Tactic for ∨E

The tactic for '∨E' : if $A \vee B$ is in F, then the problem $F \Rightarrow C$ is replaced by the problems $A :: F \Rightarrow C$, and $B :: F \Rightarrow C$.

In other words, if you already know $A \vee B$, then first assume A and see whether, with the facts you already know, you can show C, and then assume B, and see if, from this and the other facts you know, you can show C.

Tactic for →I

The tactic for '→I' : replace a problem $F \Rightarrow A \rightarrow B$ by the problem $A :: F \Rightarrow B$.

In other words, to prove a conditional $A \rightarrow B$, assume A, and try, from the facts you already know, to prove B.

Tactic for →E

The tactic for '→E': if a formula $A \rightarrow B$ is in F, then the problem $F \Rightarrow C$ is replaced by the sub-problems $F \Rightarrow A$ and $B :: F \Rightarrow C$.

In other words, if you already know $A \rightarrow B$, then see first if you can show A, then add B to what you know, and try to show C.

Tactic for ~I

The tactic for '~I' : replace the problem $F \Rightarrow \sim A$ by the problem $A :: F \Rightarrow \wedge$.

In other words, in order to show ~A, assume A, and try, with the facts you already know, to show $\wedge$.

Tactic for ~E

The tactic for '~E' : if $\sim A$ is in F then the problem $F \Rightarrow C$ is replaced by the problems $F \Rightarrow A$, and $\wedge :: F \Rightarrow C$.

In other words, if you already know $\sim A$, first try to show A, for then you can add $\wedge$ to what you know, in order to try to show C.

Tactic for DN

The tactic for 'DN' : either replace the problem $F \Rightarrow C$ by the problem $F \Rightarrow \sim\sim C$, or, where $\sim\sim A$ is in F, by $A :: F \Rightarrow C$.

In other words, either work backwards from the goal C by trying to show $\sim\sim C$, or work forwards from something of the form $\sim\sim A$ in what you know, by adding A to what you know (by DN).

Tactic for $\wedge$E

The tactic for '$\wedge$E' : if $\wedge$ is in F, then the problem $F \Rightarrow C$ is replaced by the (trivial) problem $C :: F \Rightarrow C$.

Tactic for ∀I

The tactic for '∀I' : replace the problem $F \Rightarrow (\forall x)A(x)$ by the problem $F \Rightarrow A(y)$, where x and y are variables and y does not occur free in any formula in F, or in $A(x)$.

In other words, to show the goal $(\forall x)A(x)$, try to show $A(y)$, where y does not occur free in any assumptions or in the goal. [The variable y is chosen automatically by MacLogic if alpha-conversion is on, otherwise you are asked to choose it.]

Tactic for ∀E

The tactic for '∀E' : if a formula $(\forall x)A(x)$ belongs to F, then the problem $F \Rightarrow C$ is replaced by the problem $A(t) :: F \Rightarrow C$, for some term t supplied by the user.

In other words, if you already know $(\forall x)A(x)$, add $A(t)$ to what you know (by ∀E), and continue to try to prove the goal C. Note that t is a variable, and must be free for x in $A(x)$ – if alpha-conversion is off.

Tactic for ∃I

The tactic for '∃I' : replace the problem $F \Rightarrow (\exists x)A(x)$ by the problem $F \Rightarrow A(t)$ for some term t supplied by the user.

In other words, to show $(\exists x)A(x)$, try to show $A(t)$ for some term t. Note that t is a variable, and must be free for x in $A(x)$ – if alpha-conversion is off.

Tactic for ∃E

The tactic for '∃E' : if a formula $(\exists x)A(x)$ belongs to F, then the problem $F \Rightarrow C$ is replaced by the problem $A(y) :: F \Rightarrow C$, where y does not occur free in any formula in F nor in C.

In other words, if you already know $(\exists x)A(x)$, then assume $A(y)$ for a new variable y, and then, with $A(y)$ added to the facts you know, try to show the goal C. [The variable y is chosen automatically by MacLogic if alpha-conversion is on, otherwise you are asked to choose it.]

Tactic for =I

The tactic for '=I' : regard a problem $F \Rightarrow t = t$, where t is any term, as a trivial problem. MacLogic does NOT require you to invoke this tactic.

In other words, it is trivially true that $t = t$, for any term t.

Tactic for =E

The tactic for '=E': if a formula $s = t$, where s and t are any terms, is in F, then the problem $F \Rightarrow C$ is replaced by the problems $F \Rightarrow A(s)$ and $A(t) :: F \Rightarrow C$. The formula $A(v)$ and the free variable v to be replaced by s,t are entered by the user.

In other words, if you already know that $s = t$, and can prove $A(s)$, then you may add $A(t)$ to the facts usable to prove C.

Tactic for =symm

The tactic for '=symm': if a formula $(s = t)$, where s and t are any terms, is in F, then the problem $F \Rightarrow C$ is replaced by the sub-problem $(t=s) :: F \Rightarrow C$.

In other words, if you already know that $s = t$, then you know that $t = s$. This tactic is not necessary, since it can be derived from '=I' and '=E', but is convenient.

Tactic for =trans

The tactic for '=trans': if the formulae $s = t$ and $t = u$, where s, t and u are any terms, are in F, then the problem $F \Rightarrow C$ is replaced by the sub-problem $(s = u) :: F \Rightarrow C$.

In other words, if you already know that $s = t$ and that $t = u$, then you know that $s = u$. This tactic is not necessary, since it can be derived from '=I' and '=E', but is convenient.

Tactic for Sequent Introduction

The tactic for Sequent Introduction is the tactic corresponding to the rule of Sequent Introduction (SI). It enables you to use theorems already proved, or saved in a Theorems file, in order to attain the goal. Use the … button in the Tactic choice dialog to use this tactic.

If the sequent being appealed to is of the form $A1,\ A2,\ \ldots An \vdash D$, then you will replace the problem $F \Rightarrow C$ by the new problems $F \Rightarrow A1'$, $F \Rightarrow A2'$, … , $F \Rightarrow An'$, and $D' :: F \Rightarrow C$, where the primes ' indicate that a substitution has been applied. You are prompted for the details of this substitution.

Tactic for Tautology

Use the … button in the Tactic choice dialog to choose this tactic. You are prompted to enter a sequent, which is checked for validity in propositional logic by the theorem-prover: you then have the problems of proving all its assumptions, and may add its conclusion to the fact list F of the current problem $F \Rightarrow C$. [The validity checker may in fact be a little more generous, in allowing some valid sequents of first-order logic to be entered.]

Tactic for Definition

'Definition' is the tactic that allows the expansion or contraction of the definition of '↔' or of '~'. Note that the tactic will only rewrite whole formulae of the form $A \leftrightarrow B$ [into the form $(A \rightarrow B)\&(B \rightarrow A)$] or $\sim A$ [into the form $A \rightarrow \Lambda$] where A, B are any wffs. Use the … button in the "Tactic choice" dialog to obtain the use of this tactic.

Tactic for □I

The tactic for '□I' : replace a problem $F \Rightarrow \Box A$ by the problem $F1 \Rightarrow A$, where $F1$ contains all the appropriately modal formulae in F.

In other words, to prove that A is necessary use the facts that you know and which the logic counts as modal, to prove A.

Tactic for □E

The tactic for '□E': if a formula $\Box A$ is in F, then the problem $F \Rightarrow C$ is replaced by the sub-problem $A :: F \Rightarrow C$.

In other words, if you already know that A is necessary then you know A is true.

Tactic for ◊I

The tactic for '◊I' : replace a problem F ⇒ ◊A by the problem F ⇒ A.

In other words, to prove that A is possible use the facts that you know to prove A is true.

Tactic for ◊E

The tactic for '◊E': if a formula ◊A is in F, and C is appropriately co-modal, then the problem F ⇒ C is replaced by the sub-problem A :: F1 ⇒ C where F1 contains all the appropriately modal formulae in F.

In other words, if you already know that A is possible then you may use A, and other facts that you know and that the logic permits to be used, to derive certain appropriately co-modal conclusions.

Tactic for ⥽I

The tactic for '⥽I' : replace a problem F ⇒ A⥽B by the problem A :: F1 ⇒ B, where F1 contains all the appropriately modal formulae of F.

In other words, to prove a strict conditional A ⥽ B, assume A, and try, with those facts that you know and that the logic permits to be used, to prove B.

Tactic for ⥽E

The tactic for '⥽E': if a formula A ⥽ B is in F, then the problem F ⇒ C is replaced by the sub-problems F ⇒ A and B :: F ⇒ C.

In other words, if you already know A ⥽ B, then see first if you can show A, then that once you add B to what you have, you can then show C.

Thinning

'Thin' is the tactic that allows the removal of formulae from the list F in a problem F ⇒ C. You may find it easier to see how to get from the facts you know to the goal, if you remove from the fact list F, any formulae which you will not need. Use the … button in the "Tactic choice" dialog to use Thin.

But DO NOT remove formulae which you really need (the validity checker will warn you if you do - if it is switched on!).

Auto-Thin

'Auto-Thin' automatically removes formulae from the fact list F when certain tactics are used. For example, using the tactic for '&E', when A and B are added to the fact-list F, the formula A & B is removed from F. This feature can be switched on and off by means of a check button on the "Tactic choice" dialog.

Cut

'Cut' is the tactic corresponding to Gentzen’s CUT rule: it allows a problem to be divided into two parts, as follows. Suppose the problem is F ⇒ C, and one applies 'Cut'. One is prompted for a 'Cut formula', A; the problem is then replaced by the two problems F ⇒ A and A :: F ⇒ C. The tactic can be avoided entirely in the non-modal systems, in the sense that any proof using the 'Cut' rule can be mechanically transformed into a cut-free proof. (This is Gentzen’s “Hauptsatz”, or Cut Elimination Theorem). In the modal systems, Cuts may be necessary (depending on how you interpret the restrictions on the rules and tactics for modal operators).

5.5 Differences between the various logics

Classical logic we see as an extension of intuitionistic logic, which is in turn an extension of minimal logic.

Minimal logic lacks the rule ΛE (Absurdity Elimination, *alias* “ex falso quodlibet”) of intuitionistic logic

Classical logic replaces this rule by **DN**, that of double negation (from which ΛE is derivable).

The modal rules can be added to each of the three basic kinds of logic: they are formulated to be independent of a classical or intuitionistic bias.

In constructing proofs, the main difference is that in intuitionistic logic, one has to prove a formula of the kind $A \vee B$ directly by choosing (perhaps after all other approaches have failed) one of A, B and proving it: in classical logic you can use **DN**, and try proving $\sim\sim(A \vee B)$ instead. Similarly, in intuitionistic logic you must prove $(\exists x)A(x)$ by (if other approaches fail) choosing a term t for which you can find a proof of $A(t)$: but in classical logic, you can try proving $\sim\sim(Ex)A(x)$ instead. This makes the search for proofs in intuitionistic logic more straightforward, in that the tactic for double negation is not available.

On the other hand, the mechanical check of validity in classical logic can be done much faster (especially for complex problems) than for the other two logics. However, the calculus LK used for this purpose in classical logic is rather different from the one in which proofs are traditionally presented.

5.6 Menus

MacLogic has the following menus, in addition to the usual menu: **File , Edit , Logic, Problem, Options, Windows , Help**. The purpose of this section is to describe the facilities available from each menu in turn.

The same information is available on-line as described under the item, **Menus**, in the **Help** menu. It is kept in the file “Menu Help” in a special format: provided the format is retained, the file may be modified with a word processor, or even with MacLogic. (Keep a back-up copy of the original!)

For simplicity, the same format is retained in the following, which consists of a menu name, the item name, and then details of the menu item. These details are, in the on-line version (but not in this printed version), stored as a comment, surrounded by /* ... */.

Note that the first two menu names are “File ”, “Edit ” : the spaces are important. Similarly for the menu name “Windows ”. The character “…” in menu item names is always a single character, Option Semicolon, rather than three separate dots.

Only edit the bits of the file in between the comment marks /*, */, and be concise: if the file is too big (above 32K) then MacLogic will fail to load it properly. The file “Menu Help” must be kept in the “HELP” folder, which should be in the same folder as MacLogic itself.

The menu contains an item, **About MacLogic…**, which can be used to find out the date on which your version of MacLogic was constructed, and to reset the “evaluation space” used by MacLogic. If you run out of space, while running MacLogic, it may be a good idea to change this. Try to ensure that the evaluation space is about the same as the free space. (The technically minded may wish to know that the former is stack space, the latter is heap space.)

5.6.1 File menu

File Open text file…

Open a text file, creating an editable window of the same name as the file. You will be prompted to identify the file.

The file’s name should not be that of any item in the “Windows” menu.

File Load theorems…

Load theorems from an external 'code' file in order to be used for “Sequent Introduction”. You will be prompted to identify the file. They can either replace or supplement those already in memory (as visible in the “Theorems” window).

File Load library problems…

Load a library of problems to be solved. The menu item leads into a sub-menu of choices, for loading a text file, a coded file, or the front window. The problems can either replace or supplement those already contained in the Library.

File Remove library problems…

Displays a scrolling menu of problems, as in the current library. To remove one or more select (maybe using shift-click) those to be removed, and press “Ok”. You will then want to save these library problems to a file before quitting MacLogic.

File Save to text file…

Save the front window (if any) to a text file. You can choose a new name, so long as it is not the name of another item in the “Windows” menu.

File Save theorems…

Save theorems (i.e. just proven sequents, not the proofs) for use in future sessions by “Sequent Introduction”.

The theorems are saved not as text but as code, to discourage you from “proving” theorems with a word-processor.

File Save library problems…

Save all problems currently in memory to either a text file or a code file. (Use code files for speed: use text only if you want to look at or edit the problems with a word-processor.)

The file’s name should not be that of any item in the “Windows” menu.

File Page setup…

Set up the page size etc. for printing. Note that page size should normally be A4 for use with the Apple LaserWriter™.

File Print visible windows…

Print all the visible windows.

File Reset

This may be necessary in extreme circumstances. “Reset” enables various disabled menus and menu items. (These are disabled during a proof to prevent cheating.)

Don’t use it unless you must!

File Quit

Quit from MacLogic to go back to the desktop. If you confirm this, you are prompted to save any results proved, any changes made to editable windows, and maybe the current menu settings.

5.6.2 Edit menu

Edit Undo

Undo the last edit action on the front window or dialog box.

Edit Cut

Cut selected text and put on clipboard — this can then be pasted elsewhere, if desired.

Note that some windows are “non-editable”: but, dialogs, the “Jotter” and any windows you create yourself are “editable” and so can be cut from.

Edit Copy

Copy selected text onto clipboard, without removing it from the screen. This can be done from any window or dialog box.

Edit Paste

Paste contents of clipboard into current cursor position, replacing whatever is currently selected.

Note that some windows are “non-editable”: but, dialogs, the Jotter and windows you create yourself are “editable”, and so can be pasted into.

Edit Clear

Clear selected text. Text is NOT put on clipboard.

Note that some windows are “non-editable”: but, dialogs, the Jotter and windows you create yourself are “editable”, and so can be cleared from.

Edit Balance

Find the bracket that balances with the selected one. Try repeating the action, to find the next outermost pair of brackets.

Edit Select all

Select all the text in the front window. You can then “Copy” this text, and maybe “Cut”, etc as well.

5.6.3 Logic menu

Logic Minimal

TICKED: MacLogic will use Minimal logic as the basic system.

UNTICKED: Either Intuitionistic or Classical logic will be the basic system.

Logic Intuitionistic

TICKED: MacLogic will use Intuitionistic logic as the basic system.

UNTICKED: Either Minimal or Classical logic will be the basic system.

Logic Classical

TICKED: MacLogic will use Classical logic as the basic system.

UNTICKED: Either Minimal or Intuitionistic logic will be the basic system.

Logic Quantifiers

TICKED: MacLogic will extend the basic system with rules or tactics for quantifiers.

UNTICKED: no rules or tactics for quantifiers.

Logic Equality

TICKED: MacLogic will extend the basic system with rules or tactics for equality.

UNTICKED: no rules or tactics for equality.

Logic Modal S4

TICKED: MacLogic will extend the basic system with rules or tactics for the Modal logic S4.

UNTICKED: no rules or tactics for the Modal logic S4.

Only one of Modal S4 and Modal S5 may be ticked.

Logic Modal S5

TICKED: MacLogic will extend the basic system with rules or tactics for the modal logic S5.

UNTICKED: no rules or tactics for the modal logic S5.

Only one of Modal S4 and Modal S5 may be ticked.

5.6.4 Problem menu

Problem Dialog…

Prompts you to enter a problem to be solved.

Problem Front Window

If text in the front window is selected, MacLogic tries to parse that either as a sequent or as a formula. If successful, it will allow you to start solving that problem.

Problem Library

Prompts you to select a problem from the library.

If no problems are loaded, then you will be asked to find a problem file. (On startup, MacLogic looks for a code file called “MacLogic Problems” and pre-loads this if it is there.)

Problem Test run

Runs through the library problems, reporting which can be proved in the current logic with the current validity checker setting. If “Keeping as theorem” is enabled, then you may save propositional results as theorems, to build up a theorems database.

5.6.5 Options menu

Options Increase type size

Increases the type size of all windows to 12pt.

Options Decrease type size

Decreases the type size of all windows to 9pt.

Options Show keypad menu

Adds a menu “Keypad”, with submenus: this can be used to get text into a window without using the keyboard at all. Especially useful if you have some tear-off menu software.

Options Hide keypad menu

Hides the menu “Keypad”.

Options Set validity checker…

MacLogic can think about problems that you try to solve, and may warn you of attempts to solve any that it can’t itself solve. Use this menu item to control how much of this MacLogic should do.

Use the “ATP” item in the “Help” menu for more details.

Options Alpha conversion

TICKED: MacLogic will treat formulae which differ only in the names of bound variables as identical.

UNTICKED: MacLogic will treat formulae which differ in the names of bound variables as different.

Options Delta conversion

TICKED: MacLogic will implicitly use the definitions

$$\sim A =_{\mathrm{def}} A \rightarrow \Lambda$$
$$A \leftrightarrow B =_{\mathrm{def}} (A \rightarrow B) \& (B \rightarrow A).$$

UNTICKED: MacLogic will force you to expand these definitions explicitly.

Options Lots of parentheses

TICKED: When displaying formulae, MacLogic will use lots of parentheses.

UNTICKED: When displaying formulae, MacLogic will use as few parentheses as possible.

(Both kinds of syntax are allowed when formulae are being read by MacLogic.)

Options Saving to file…

TICKED: Completed proofs and derivations (but not necessarily the results) will be saved to a text file as you go along.

UNTICKED: Completed proofs and derivations are not saved to a text file.

Options Saving proofs to window

TICKED: Completed proofs are copied to the “Previous Proofs” window, to be viewed later in the current session.

UNTICKED: Completed proofs are not copied to the “Previous Proofs” window.

Options Saving proofs and derivations to window

TICKED: Completed proofs and derivations are copied to the “Previous Proofs and Derivations” window, to be viewed later in the current session.

UNTICKED: Completed proofs and derivations are not copied to the “Previous Proofs and Derivations” window.

Options Keeping as theorem

TICKED: Propositional results (i.e. assumptions and conclusion) are remembered, for use in the current session with “Sequent Introduction”. They can be viewed in the “Theorems” window.

UNTICKED: Theorems are not remembered.

Options Checking

TICKED: Proofs will be checked in a “bottom-up” style.

UNTICKED: Derivations are to be constructed in a “top-down” style: when complete, the corresponding natural deduction proof will (if you wish) be displayed.

Options Constructing

TICKED: Derivations are to be constructed in a “top-down” style: when complete, the corresponding natural deduction proof will (if you wish) be displayed.

UNTICKED: Proofs will be checked in a “bottom-up” style.

5.6.6 Windows menu

Windows Create…

Create a new (editable) window.

You must choose a name distinct from that of any item in the “Windows” menu.

Windows Refresh

Force the “Derivation” window to be redrawn, if, after being covered temporarily by a dialog window, it isn’t properly refreshed by the Macintosh system.

Windows Toggle Bold

If the front window is a text window, then it is put into (or taken out of) Bold face. This feature is intended for those using MacLogic in the class room with an LCD panel and an overhead projector.

Windows Hide

Hide the front window. (You could use its “close box” instead.)

Windows Kill…

Kills the front window, if it is one you have created yourself.

You can cancel if you change your mind.

Windows Previous Proofs and Derivations

Make the “Previous Proofs and Derivations” window the front window, so that you can see the work you have done (and saved in this window) in the current session.

Windows Previous Proofs

Make the “Previous Proofs” window the front window, so that you can see the proofs you have done (and saved in this window) in the current session.

Windows Theorems

Make the “Theorems” window the front window. This window shows all the theorems which may be used in “Sequent Introduction”.

Theorems can be loaded from a file if required: see the “File” menu.

Windows Derivation

Make the “Derivation” window (i.e. the window that is used to show the derivation in Construct mode) the front window.

Windows Current Problem

Make the “Current Problem” window (i.e. the window that is used to show the current problem in Construct mode) the front window.

Windows Proof

Make the “Proof” window the front window.

Windows Jotter

Make the “Jotter” window the front window.

This is a window for you to use as you wish: for example, it’s handy for typing complex problems into, getting the brackets right (using Edit/Balance), etc. It is editable, but can’t be killed.

5.6.7 Help menu

Help Syntax

Help about the syntax, e.g. which symbols are used, etc. (The information is held in a text file called “Syntax Help” , which must be in the HELP folder in the same folder as MacLogic.)

Help Rules

Help about the inference rules, for use in Check mode. (The information is held in a text file called “Rules Help” , which must be kept in the HELP folder in the same folder as MacLogic.)

Help Tactics

Help about the tactics, for use in Construct mode. (The information is held in a text file called “Tactics Help” , which must be kept in the HELP folder in the same folder as MacLogic.)

Help ATP

Exposes a window of information about the Automatic Theorem Proving component of MacLogic.

Help Menus

All other menu items will now give help about themselves. Switch help off by reselecting “Help/Menus.”

Help Valid

Checks whether a problem is solvable in first-order logic, without using the modal or equality rules.

The problem can be entered either via a dialog or as the text selected in the front window.

Help Identical

Lets you check whether two formulae are identical, i.e. convertible according to whatever conversion rules (alpha-conversion and delta-conversion) are enabled in the Options menu.

Help Modal

Checks whether a formula is appropriately modal for the modal systems S4 and S5.

(See Help/Syntax for the definition.)

Help Co-modal

Checks whether a formula is co-modal for the modal systems S4 and S5.

(See Help/Syntax for the definition.)

Help Font problems

If you have problems when MacLogic dialogs and menus don't show the logic symbols properly, this will tell you what to do.

Help Interruption

This tells you how to interrupt MacLogic: use Command Period. It is useful if you see the validity checker getting stuck, or you accidentally try to read some junk as a formula.

Help Memory problems

If you have problems with MacLogic running out of memory, this will tell you what to do.

6. Acknowledgements, disclaimer, availability and licence arrangements

MacLogic was created at the University of St Andrews, Fife, Scotland, under the **MALT** (Machine Assisted Logic Teaching) Project, a project for the encouragement of computerised logic teaching, funded from 1987 to 1989 by the United Kingdom's University Grants Committee through its *Computers in Teaching Initiative*.

Those with some direct responsibility for the program are:

Roy Dyckhoff, Bob Hale, Neil Leslie, Brenda Rapley, *and* Stephen Read

MacLogic was awarded First Prize in the 1989 Philosophy Software Competition, organised by the Philosophy Documentation Center, Bowling Green State University, Ohio, USA.

DISCLAIMER

MacLogic was developed using LPA[12] MacPROLOG™, and includes the MacPROLOG Runtime Software, all copyright and industrial rights therein being owned by LPA Ltd. No warranty is given as to the merchantable quality or fitness for any particular purpose of this software, and no liabilities of any kind are accepted either by LPA, the MALT project, St Andrews University, the distributors, or anyone else whatsoever. In particular, no guarantee is given that the implementation of any particular logic is consistent and complete.

MacLogic is distributed for at a non-commercial charge on condition that comments are reported to the authors. Licences to use it are available for purchase: for details write (or send e-mail) to the address below. A demonstration copy is available by anonymous file transfer.

No charge is made to students at an institution with a site licence. They are free to use MacLogic on the institution's machines, and on their own, on condition that they use it only for learning about logic.

Contact:

Dr Roy Dyckhoff,
Machine Assisted Logic Teaching Project,
Computational Science Division,
University of St Andrews,
St Andrews, Fife, SCOTLAND, UK.

☎ +44-334-76161 ext^{n} 8134/8262
FAX: +44-334-77068

e-mail: rd@dcs.st-and.ac.uk

---

[12] Logic Programming Associates Ltd, Studio 4, Royal Victoria Patriotic Building, Trinity Road, London, SW18 3SX, England, tel 081 871 2016, fax 081 874 1449.