A Brief Guide to *MacLogic*

Graeme Forbes

MacLogic is a program for the Apple Macintosh computer which implements Gentzen's system of natural deduction NK as well as other logical systems. It functions in two modes, either (a) as a proof-checker, in which the user types in a proof line by line and the program checks each line, detecting errors and displaying brief correction messages where appropriate, or (b) as a proof-constructor, in which the user dictates the strategy to be pursued and the program builds the proof. In the following quick guide to *MacLogic*, basic familiarity with the Mac is assumed; if you've never used a Mac before, play around with one and follow Apple's introductory documentation before starting on *MacLogic*. This guide will be enough to get someone who knows his or her way around the Mac up and running with the program. However, the distribution package for *MacLogic* contains a much fuller and more detailed manual (as a *Microsoft Word* file) which the serious user should print out and study (or see http://www.dcs.st-and.ac.uk/~rd/logic/mac/docs/).

MacLogic is written in a programming language called *Prolog* (for 'programming in logic'). Different versions of Prolog work on different generations of Macs and with different system settings. But unless you have a very old Mac, you should be using the latest version of *MacLogic*, 3.4.1, which you can download at http://www.dcs.st-and.ac.uk/~rd/logic/mac/. *MacLogic* requires OS9 to run, so if your Mac runs under OSX, it will launch in Classic or not launch at all. If it doesn't launch at all it's likely you don't have Classic installed. There are two cases: (A) you have one of the last Macs based on IBM chips, the G4 or G5. In this case you need to insert the DVD that came with your Mac, find the Classic Installer, and install Classic. (B) Your Mac is one of the newest, and has an Intel chip. In this case Classic cannot be installed, and you must

run *MacLogic* inside a program that simulates an older Mac. An easy-to-use package that allows this can be found at http://fitelson.org/maclogic.htm. The simulation program is called Basilisk, and there are versions of it for Windows, Linux, BeOS and other platforms; so *MacLogic* can be run essentially on every type of personal computer. (If your Mac hails from the mid-90s or earlier, please email me for help; forbes@tulane.edu)

Before proceeding any further, you should be aware of a potentially confusing difference in terminology between *MacLogic* and *Modern Logic*. In *Modern Logic*, following Lemmon, I use the term 'premise' to mean 'premise of a sequent', that is, a formula on the left of the turnstile '⊢' of the sequent. But in *MacLogic* the word 'premise' means 'premise for the application of a rule of inference'. For example, if we apply &I to two lines j and k to infer a conjunction 'A & B' at a line m, the conjuncts 'A' and 'B', or the lines j and k, are said to be the *premises for* that application of &I. *MacLogic* uses the word 'Assumption' for what are called assumptions in *Modern Logic* and *also* for what are called premises there: the program views what *Modern Logic* calls premises simply as assumptions which need not be discharged.

In order to use *MacLogic* one has to be able to type the logical symbols. The program comes with two logic fonts, Detroit and Konstanz. (For those familiar with the Mac, Detroit is a Chicago lookalike which contains the logic symbols and is used in the program's menus in place of Chicago. Konstanz is a Geneva-based text font with logic symbols, and is the font in which the program displays proofs.) If you are using *MacLogic* on a computer on which it has not been run before, you will probably need to install the fonts. Consult your Apple documentation to determine whether you should install only bitmap fonts, or TrueType fonts, or bitmap and type 1 fonts.

Logic symbols are produced by special key combinations. Beneath the Shift key on a Mac keyboard there is a smaller key called the Option key. A logic symbol is typically produced by pressing the Option and Shift keys together and then pressing a character key so that all three keys are depressed simultaneously. The key combinations for the symbols are displayed in the table on the following page. In the USA release of System 7, Apple remapped the keyboard, making some logic symbols harder to produce. You might be able to restore the System 6 key-

board using the Keyboard control panel. You may also want to obtain

Symbol	*System 6 Keystroke*	*System 7, 8, 9, Keystroke*
~	Option-Shift-L	Option-Shift-L
∨	Option-Shift-D	Option-Shift-D
→	Option-Shift-Y	Option-Shift-Y
↔	Option-Shift-S	Option-Shift-S
⅄	Option-Shift-F	Option-Shift-F
∀	Option-Shift-U	Option-U and then Shift-E
∃	Option-Shift-E	Option-Shift-R
□	Option-Shift-N	Option-Shift-I
◇	Option-Shift-V	Option-Shift-V

Table 1: Keystrokes for Logic Symbols in Konstanz

the freeware control panel *PopChar*, which displays a font's character set on screen, tells you what keystroke will produce any character you select, and will pop that character into your text at the current cursor position. (The most recent versions of *MacLogic*, 3.*x* and later, allow certain other ways of creating symbols – see the Advice window that is displayed on startup.)

❑ Proof-Checking

Launch *MacLogic* by double-clicking the program icon (the hand drawing '∃'). You will be presented with some messages which you should read and then dismiss by clicking 'Ok'. Pressing the Return key is the same as clicking 'Ok', or more generally, it is the same as clicking the button with the thick border (if there is one), the *default* button, in any dialog box. You will now be presented with the following menu bar at the top of your screen:

File Edit Logic Problem Options Windows Help

The Apple menu (the one which appears by clicking on the) allows you to display the information box which also appears when you launch *MacLogic* (this is useful if you want to reset the evaluation space in *MacLogic* 2.5 or earlier) and then underneath, the names of the various 'accessories' which are installed in your system (these appear no matter what program is running). You should take a moment to explore the other seven menus, which belong to *MacLogic* itself. Under the **File** menu are various options, none of which need concern you at the moment. The **Edit** menu contains the usual Mac operations and a special **Balance** command. If you highlight a parenthesis in a formula and choose **Balance**, the program will find the matching parenthesis (if there is one) in the formula. This is useful if the program has told you that you have typed a non-wff. The **Logic** menu offers a choice of various logical systems. Make sure **Classical** is checked. The other menu to look at before beginning is the **Problem** menu. Make sure **Delta Conversion** is *not* checked and **Checking** *is* checked. The program defaults to **Delta Conversion** off, but **Constructing** rather than **Checking** may be activated (**Alpha Conversion** is irrelevant at this point but later you will want it off). *MacLogic* 2.5 or earlier will let you save the settings you select, but 3.*x* will not and you will have to reset them each time you launch the program. Remember to set **Classical** in the **Logic** menu**.**

To demonstrate a simple proof-check, we will check this proof:

Example 1: Show $P \rightarrow (Q \rightarrow R) \vdash (P \rightarrow Q) \rightarrow (P \rightarrow R)$.

1	(1)	$P \rightarrow (Q \rightarrow R)$	Premise
2	(2)	$P \rightarrow Q$	Assumption
3	(3)	P	Assumption
1,3	(4)	$Q \rightarrow R$	1,3 →E
2,3	(5)	Q	2,3 →E
1,2,3	(6)	R	4,5 →E
1,2	(7)	$P \rightarrow R$	3,6 →I
1	(8)	$(P \rightarrow Q) \rightarrow (P \rightarrow R)$	2,7 →I

To derive '$(P \rightarrow Q) \rightarrow (P \rightarrow R)$' we expect to use →I at the last line, requiring that we assume the antecedent '$P \rightarrow Q$' (line 2) and derive the consequent '$P \rightarrow R$' (line 7). Since '$P \rightarrow R$' is another conditional, we would

expect to derive it by →I too, so we assume its antecedent 'P' (line 3) and derive its consequent 'R' (line (6).

To check that this is a correct proof we enter it line by line into *MacLogic*. First we have to enter the problem itself. From the **Problem** menu choose **Dialog**, aand type in the premises and conclusion. Clicking **Ok** will cause the screen to redraw and two new windows will appear, the Proof window and the Next Line entry window, as shown in Figure 1:

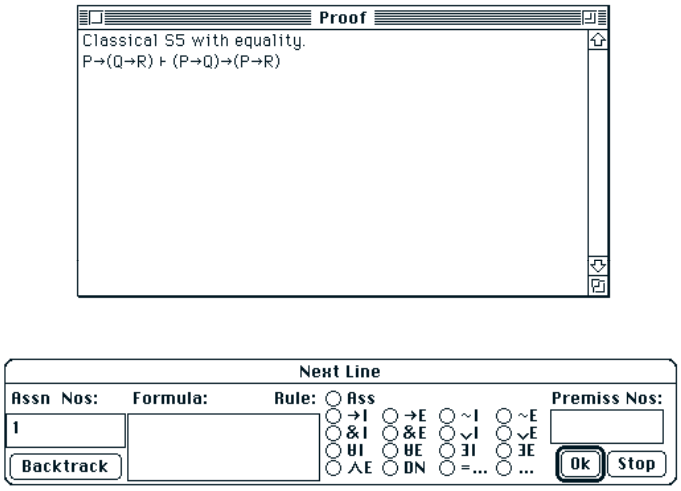


Figure 1

Click in the **Formula:** box and type the formula 'P → (Q → R)' (without the quotes; you may be able to type '->' to get '→'). Then click the radio button next to **Ass** and then click **Ok** or hit Return. The first line of the proof will be displayed in the **Proof** window and the **Assn. Nos:** and **Formula:** boxes will clear.

We can now enter the second line of the proof. Click in the **Assn. Nos:** box and type '2', Tab or click in the **Formula:** box and type 'P → Q' (*MacLogic* will not insist on outer parentheses), click the radio button next to **Ass**, then click **Ok** or hit Return. The second line of the proof will be displayed in the **Proof** window and the **Assn. Nos:** and **Formula:** boxes will clear. Similarly, enter the third line. There are now three lines in the **Proof** window, all labelled **Ass**—recall that *MacLogic* uses the same label for premises and assumptions.

Next, we enter the first line in this proof which is obtained by applying a rule of inference to previous lines. Type '1,3' in the **Assn. Nos:** box, 'Q → R' in the **Formula:** box, click in the radio button for →E, and then, in the **Premiss Nos:** box, type in the line numbers to which the rule of →E is being applied in this step, that is, type in '1,3'. Remember that *MacLogic* calls the lines to which a rule is being applied the *premises for* that application of the rule. That is why the **Premiss Nos:** box is so-called. If you like, instead of typing '1,3' in the **Premiss Nos:** box, type '3,1', then click **Ok**. You will get an error message to the effect that the premises are in the wrong order. *MacLogic* is fussy about the order in which you type the line numbers to which a rule is being applied. The general principle it follows for an Elimination rule is this: the number of the line that contains the formula with the connective occurrence being eliminated is entered first, then the number of any other line which is involved. The line with the connective occurrence being eliminated is called the *major premise* for that application of the E-rule, and the other line(s), if any, the *minor premise(s)*.

Once you have '1,3' in the **Premiss Nos:** box, type **Ok**. In the same way, enter lines 5 and 6 of the proof, remembering that in the **Assn. Nos:** box for line 6, you have to type '1,2,3'. The **Proof** window now looks like this:

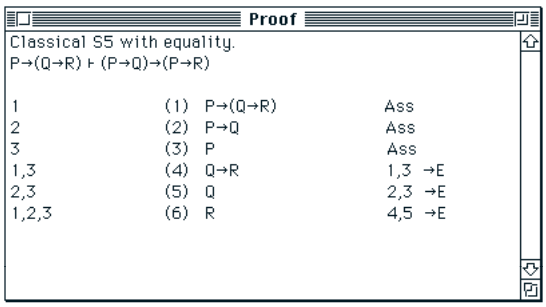


Figure 2

Now enter the remaining lines of the proof. If at any point you make a technical mistake in entering a line, for example if you forget to discharge assumptions and leave too many numbers in the **Assn. Nos:** box, *MacLogic* will catch it when you click **Ok**. If you enter a line which is technically correct but which you decide you don't want when you

see it displayed in the **Proof** window, you can use **Backtrack** to go back to the line and change it. After you have typed in the last line and clicked **Ok**, you will get a message of congratulations.

Your next step will typically be one of the following actions: going to the **File** menu and choosing either **Save to text file** or **Print Visible Windows**... or **Quit**, or going to the **Problem** menu and choosing either **Dialog**... or **Library**.

- **Save to text file** brings up a dialog box with a default file name that you can change. Click the **Save** button. You can open this file later with a word-processor and print it, but only if you have the font **Konstanz** installed (see next item). However, a more efficient way of generating a single file containing all the proofs you generate in a *MacLogic* session is to have a word-processor running simultaneously. Then at the conclusion of a proof the contents of the Proof window can be copied and pasted into the word processor file.

- **Print Visible Windows**... will print out the completed proof if your Mac is connected to a printer. The command is quite literal—*all* windows any parts of which are visible on the screen will be printed, one per page. If you only want to print out the proof, close all other windows first.

- **Quit** allows you to leave *MacLogic* and returns you to the Finder. If you changed settings on the **Logic** menu or elsewhere from what they were previously, you will be asked (2.5 or earlier) if you want to save the new settings. You may want to do this if you are running *MacLogic* on your own machine and would like to have the current settings in force next time you use the program. Click **Yes** then navigate to the folder where the Settings file is stored, click **Save** and **Yes** when asked if you want to replace the old Settings file.

- **Dialog**... allows you to start over checking another proof, one for a problem you enter yourself rather than select from the library. Use of the **Problem Entry** window which **Dialog**... calls up is described in the next section. Your current proof will be stored in a window called **Previous Proofs** and can be consulted at any point by choosing this window from the **Windows** menu. Note that completed proofs will only be stored in this window if at the start of the session you

checked **Saving proofs to window** in the **Options** menu. If all you are doing is checking proofs you have already written out, there is no need to save them.

- As we have seen, **Library** saves you the effort of typing the premises and conclusion of a sequent. *MacLogic* may come preloaded with a library of problems, and the fitelson.org packages contain most of the *Modern Logic* problems as text files that you can load by choosing **Load library problems** from the **File** menu. This presents you with a pop-up submenu from which you can choose **From text file**… and navigate to the file you want. On the *MacLogic* distribution disk the other Problem files are in a folder called 'Problem files'.

In the **Next Line** window, one of the radio buttons is labelled '…' Clicking on this button in *MacLogic* 2.5 or earlier allows one to use Df, SI or a rule called 'Tautology' (SI is not supported in *MacLogic* 3.*x*, and users should employ Tautology instead). Choosing SI from the dialog box brings up a scrolling list of sequents and theorems, and you can double-click on the one you want to use. However, the list is less comprehensive than that in *Modern Logic* (p. 123). To use a sequent in SI that is not in *MacLogic*'s list, click the **Tautology** button. If the formula you enter in the **Formula:** box really is a semantic consequence of the formulae at the line(s) whose number(s) you enter in the **Premiss Nos.** box, the appropriate new line will be added to your proof. At least initially, you should restrict yourself to the sequents (a)–(r) on p. 123 of *Modern Logic* for uses of SI *via* the **Tautology** button. A file with these sequents ready to load is part of the packages from fitelson.org

❑ Constructing Proofs

Launch *MacLogic* and make sure that **Constructing** is checked in the **Options** menu. If you are currently using *MacLogic* in the **Checking** mode, simply finish your proof, or click **Stop** in the line-entry dialog, then go to **Options** and check **Constructing**. Next, choose either **Dialog**… or **Library** from the **Problem** menu. For this example, we will suppose that you choose **Dialog**, in which case the following Problem Entry

box appears on the screen:

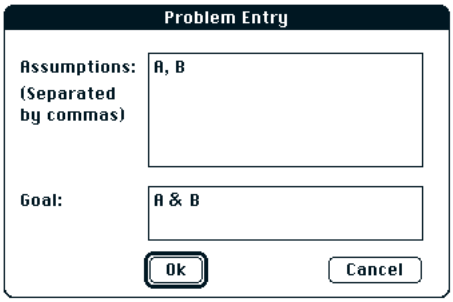


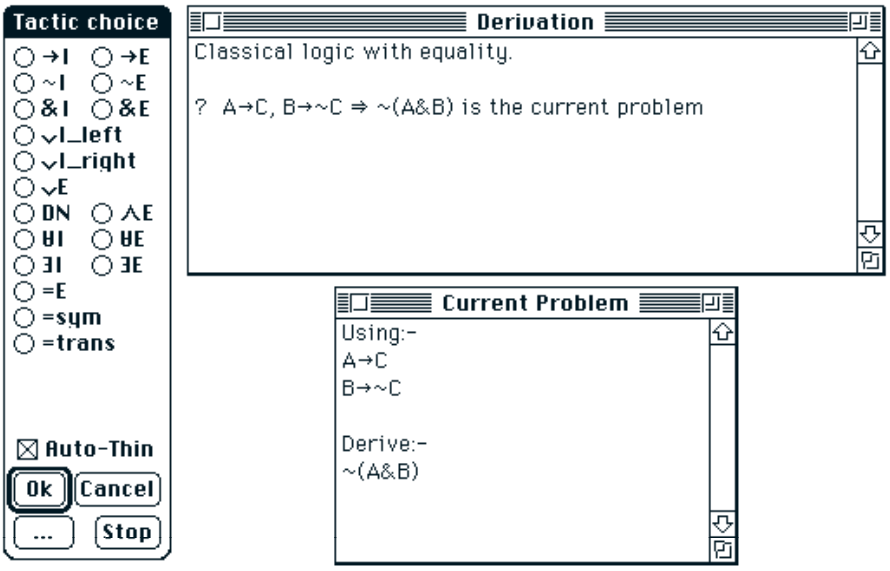
Figure 3

Probably the 'A, B' in the top box is highlighted (white text on a black background) in which case the first keystroke you type will delete it. Otherwise, click in front and then backspace. After entering the premises, tab into the Goal box and type the conclusion. In this illustration, we will construct a proof of the following problem:

Example 2: Show A → C, B → ~C ⊢ ~(A & B)

1	(1)	A → C	Premise	
2	(2)	B → ~C	Premise	
3	(3)	A & B	Assumption	
3	(4)	A	3 &E	
1,3	(5)	C	1,4 →E	
3	(6)	B	Assumption	
2,3	(7)	~C	2,6 →E	
1,2,3	(8)	⅄	7,5 ~E	
1,2	(9)	~(A & B)	3,8 ~I	◆

Once you have entered the premises into the Assumptions box and the conclusion into the Goal box, click **Ok**. You are now presented with a very different-looking display, as illustrated on the following page. In **Constructing** mode, we do not tell the program what the lines of the proof are but only what tactics (rule applications) are to be used in constructing the proof. In Example 2, the overall goal, and so the *current problem*, is to derive '~(A & B)' from the two premises. We would expect to obtain '~(A & B)' by ~I, so we click the ~I radio button in the **Tactic**



Choice box, then click **Ok** or hit return. The contents of both the **Derivation** window and the **Current Problem** window will change. The **Derivation** window may be ignored, but the contents of the **Current Problem** window are what guide us through the proof-construction. As we know, when we decide to use ~I to obtain a formula of the form ⌜~p⌝ we assume p and try to derive '∧'. So in this example, when writing the proof out, we assume 'A & B' at line 3 and try to derive '∧'. Inspecting the **Current Problem** window we see that this in effect is what has happened in it; 'A & B' has been added to the formulae which we can use and the formula to be derived has changed from '~(A & B)' to '∧'.

In order to obtain '∧', we have to apply &E, then use →E twice, then ~E. Clicking the radio button for &E and then clicking **Ok** updates the **Current Problem** window by replacing the assumption 'A & B' with the result of applying &E to it, viz. the two separate formulae 'A' and 'B'. Then we click the radio button for →E and **Ok**. Since there are two conditionals in the list of formulae we may use at this point, a dialog box asks us to select one. If the highlighted conditional is the one you want, click **Ok**, otherwise double-click the conditional you want. When →E is applied, the chosen conditional is removed from the Using:– list, along with its antecedent, and in their place the consequent appears (this has the side effect of eliminating the need to choose a conditional when we

apply →E the second time). So after two applications of →E, the Using:– list now contains 'C' and '~C'. The three applications of elimination rules have not affected the formula to be derived, which is still Absurdity, and with 'C' and '~C' in the Using:– list, we can choose ~E to finish the proof. The offer to display the proof we have just constructed should be accepted.

This method of constructing proofs requires no typing of formulae after the problem has been entered, but requires us to be able to work out how the proof should go. In our previous example we see that the proof will finish with an application of ~I, so choosing the tactic for ~I is our first step. We continue reasoning backwards from the bottom of the proof in this manner until there are no more introduction rules to be applied. In Example 2, evidently, we arrive at this point immediately after choosing ~I. So we switch to the top of the proof and start applying elimination rules. An elimination rule can be applied only if there is an appropriate formula to apply it to among the formulae currently listed in the **Current Problem** window's "Using:–" list. 'appropriate' means that if the elimination rule is for a connective c, then there must be a formula in the Using:– list with c as its *main* connective. For example, after choosing ~I to start the proof construction, we cannot continue working up from the bottom of the proof by choosing ~E (even though we know that the second last line of the proof will be derived by ~E) because at this point *there is no formula with '~' as main connective in the Using:– list.* But as soon as '~C' appears in the Using:– list, we can choose ~E. Indeed, if we choose 'B → ~C' as the first conditional to which to apply →E, then we can choose ~E immediately thereafter. What will happen then is this. In order to apply ~E we need two formulae of the form q and $\ulcorner\sim q\urcorner$. Since '~C' is the only negative formula in the Using:– list, *MacLogic* takes your choice of ~E to mean that $\ulcorner\sim q\urcorner$ is '~C'. To use ~E we therefore need q, that is, 'C', so the formula under Derive:– in the **Current Problem** window changes from '人' to 'C'. The problem can then be concluded with another use of →E. The reader should run through the construction process for Example 2 twice, in the first pass choosing →E twice, then ~E, and in the second, choosing ~E after a first application of →E to 'B → ~C'.

As in Checking mode, SI and TI are accessed through the button with the ellipsis '…'. Choose SI (which includes TI) if the sequent is one of those in *MacLogic*'s own list (if there is such a list), otherwise choose

Tautology. In the latter case you will be presented with a box listing the goal formula and *all* the formulae which may be used at the current stage. Very likely most of these are irrelevant to the application of SI you wish to make. Deleting all but the relevant ones will produce an application of **Tautology** labelled with the correct line numbers. You may also have to type a different formula in the **Succedent formula** box. For example, the quickest proof of the problem ~(P → (Q ∨ R)) ⊢ (Q ∨ R) → P applies Neg-Imp at line 2 to the premise, so one should click the ellipsis button and choose **Tautology**, which brings up the following dialog box:

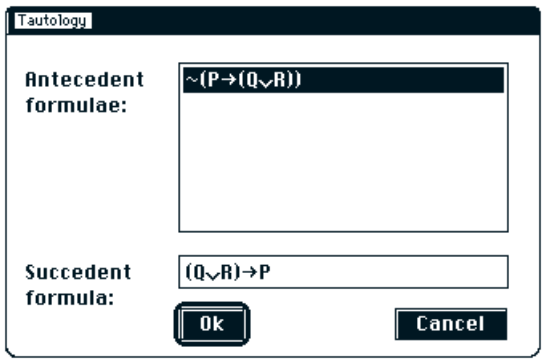


Figure 4

Though the succedent formula is indeed a semantic consequence of the antecedent formula, and *MacLogic* will allow you to click **Ok** at this point, producing a one-step proof of the entire problem, you would not expect much credit for constructing this proof! Since the intention is to apply Neg-Imp, you should tab into the **Succedent formula** box and type '(P & ~(Q ∨ R))', then click **Ok**. You will be returned to the main screen and will find that 'P & ~(Q ∨ R)' has replaced '~(P → (Q ∨ R))' in the 'Using:' list. You can now apply &E and PMI to finish the proof (if you are copying the proof into a word-processor you may want to replace the label '1 Taut' on line 2 with '1 SI (Neg-Imp)').

In other examples, the contents of the **Succedent formula** box may be right for what you have in mind, which is to obtain the goal formula from a formula p which can itself be obtained from formulae in the Using:– list. In this case delete the formulae in the **Antecedent formulae** box, enter p in their place, and click **Ok**. When you return to the

main screen you will find that the Using:– list is as before but the formula to derive has been changed to p. Now pursue whatever strategy you had in mind to derive p from the formulae you can use.

Here are two things to remember when using *MacLogic* in **Constructing** mode:

- Only choose an elimination rule for a connective *c* if *c* is the *main* connective of one of the formulae listed in the **Current Problem** window under 'Using:–'. For example, if the formula 'A & (B ∨ C)' is the only one in the list, &E is a possible tactic choice, but not ∨E, though you can choose ∨E *after* &E: &E enters 'A' and '(B ∨ C)' into the Using:– list in place of 'A & (B ∨ C)', and '∨' is the main connective of '(B ∨ C)'.

- Only choose an introduction rule for a connective *c* if *c* is *the main connective of the formula in the* ***Current problem*** *window beneath 'Derive:–'*. For example, if under 'Using:–' you have the three formulae 'A', 'B' and '(A & B) → C', and the formula to be derived is 'C', there is *no* introduction rule which can be applied at this point, since 'C' does not have a main connective. Of course, we know that we will eventually want to derive 'A & B' by &I in order to apply →E to '(A & B) → C', but if you choose &I at *this* point (i.e. when 'C' is the formula beneath 'Derive:–'), *MacLogic* will take you to think that 'C' itself can be derived by &I and will tell you that this is wrong. What we should do is choose →E. *MacLogic* will find the conditional and then look for its antecedent in the Using:– list. If it does not find the antecedent there, the formula to be derived changes from whatever is currently listed ('C' in our example) to the required antecedent. So in our example, choosing →E results in the formula to be derived changing from 'C' to 'A & B', and *now* it is legitimate to choose &I, since '&' is the main connective of the formula now listed in the **Current Problem** window under 'Derive:–'. [To familiarize yourself with this, use *MacLogic* to construct a proof of A, B, (A & B) → C ⊢ C.]

❑ Quantifiers and Identity in *MacLogic*

Using *MacLogic* to construct proofs in monadic or full first-order logic

is a straightforward extension of the procedures for sentential logic (if you want to use the identity rules, make sure **Equality** is checked in the **Logic** menu). However, to make *MacLogic*'s **Construct** mode completely compatible with *Modern Logic*, you must have **Alpha Conversion** in the **Options** menu *un*checked. This done, the following will happen when you choose a tactic for a quantifier rule.

- If the rule is ∀E or ∃I, a **Term Entry** window will appear and you will be asked to choose a term (more strictly, an individual constant) t to substitute for the variable v in the formula ϕv which your rule-application concerns. For instance, if you are applying ∀E to '(∀x)(Fx → Gx)' the following will appear:

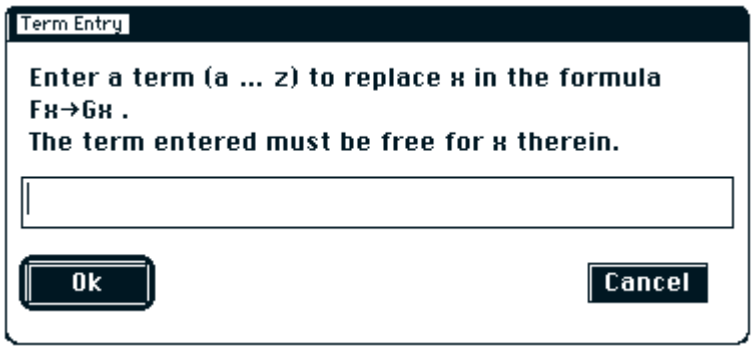


Figure 5

  You are told that the term you choose must be free for 'x' in 'Fx → Gx', or more generally, that t must be free for v in ϕv, but you may ignore this warning, since it concerns a problem that arises only when **Alpha Conversion** is on. At this point you should enter the individual constant 'a', 'b', 'c' etc. which you prefer and click **Ok**; the program will return you to your proof, and ⌜Ft → Gt⌝ will have been added to the Using:– list.

- If your chosen tactic is for ∀I or ∃E, you are presented with a scrolling window and asked to choose an individual con-

stant to replace occurrences of the variable that is relevant to your rule application.[1]

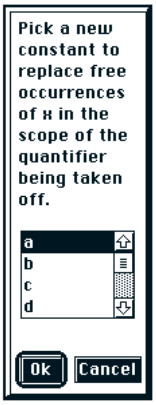


Figure 6

For example, if you are trying to deduce '(∀x)(Fx → Gx)' you may decide to aim for 'Fa → Ga', which is the formula you get if 'a' replaces the occurrences of 'x' in '(∀x)(Fx → Gx)' which become free when the '(∀x)' is removed (the syntax given in *Modern Logic* guarantees that *every* occurrence of ν in $\ulcorner(\forall\nu)\phi\nu\urcorner$ becomes free when $\ulcorner(\forall\nu)\urcorner$ is removed, but not all systems have this feature). Once you have chosen a specific individual constant, it will be removed from the list of those you may use in future applications of ∀I or ∃E. This helps keep your uses of these rules legal.

If you find that your attempts to apply quantifier rules are resisted by the validity checker, which says that you may be attempting to prove something which is invalid, it is likely that the problem is merely with the order in which you are trying to apply the rules. For example, you may have chosen the tactic for ∃I since the formula you are trying to derive is existential, but in the final proof this application of ∃I will occur within a use of ∃E. In this case you should choose the tactic for

[1] If you are presented with a window that asks you to choose a *variable* and presents you with a scrolling list in reverse alphabetical order, you have a version of *MacLogic* which has not been customized for *Modern Logic*. If you know how to use ResEdit you can customize it yourself following the instructions on p. 359 of *Modern Logic*. Otherwise simply ignore 'variable' and scroll down the list until you reach constants.

∃E before that for ∃I.

The rules for identity in *MacLogic* are =I, =E, Sym and Trans, the latter two essentially being *ad hoc* extensions of SI. Use Sym if you have, say, 'a = b' and want 'b = a', and use Trans if you have, say, 'a = b' and 'b = c' and want 'a = c'. *MacLogic* will implement =I automatically when it is needed, so there is no =I radio button.

In order to apply =E you have to tell the program which identity sentence ⌜$t = t'$⌝ you want to use (if there is more than one available), which formula you want make substitutions in, and which occurrences of t in that formula you wish to replace with t'. To illustrate how *MacLogic* handles this, suppose we have the following problem:

Example 3: Show a = b, (∀x)Fxa ⊢ Fbb.

1	(1)	a = b	Premise	
2	(2)	(∀x)Fxa	Premise	
2	(3)	Faa	2 ∀E	
1,2	(4)	Fbb	1,3 =E	◆

If, working up from the bottom of the proof, we begin by choosing the tactic for =E, the dialog below appears. We are asked to supply a formu-

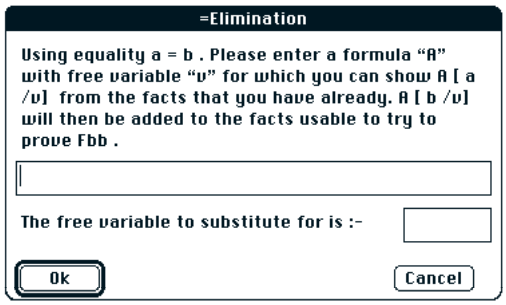


Figure 7

la with some free variable v such that the result of replacing v with 'a' is something that we can derive. We know we can derive 'Faa' so we enter 'Fxx' (or 'Fyy' etc.). We then tab down or click in the next box to specify that the free variable to replace is 'x', then we click **Ok**.

By careful choice of formula to enter in the first box we can control which occurrences of the individual constant in the minor premise of the =E are replaced when =E is applied. For instance, if we wanted 'Fab' instead of 'Fbb', we would enter, say, 'Fax' in the first box, since we can show 'Faa', and specify that it is 'x' that is to be substituted for. The program would put 'a' for that occurrence of 'x' and then replace *that occurrence* of 'a' with 'b'.

Finally, you may wish to try *MacLogic* with **Alpha Conversion** on. This allows you, when applying a quantifier elimination rule, to replace the bound variable with a free variable rather than an individual constant. For example, from '(∀x)(Fx & Gx)' one may infer 'Fx & Gx', 'Fy & Gy' etc. Generally speaking, with **Alpha Conversion** on, *MacLogic* will apply quantifier rules itself unless there is a choice to be made, in which case, as happens when **Alpha Conversion** is off, you will be asked to choose a term with which to replace the variable in the formula which results from deleting the prefixed quantifier. However, if you are going to use **Alpha Conversion**, you should understand the details of quantifier rules when free variables are allowed in proofs.

- In ∀I, the same restrictions apply to free variables as apply to individual constants. For example, if 'Fx & Gx' has been derived depending on premises and assumptions $p_1 \ldots p_n$, then '(∀x)(Fx & Gx)' (or '(∀y)(Fy & Gy)' etc.) may be inferred, depending on $p_1 \ldots p_n$, so long as there is no occurrence of 'x' free in any of the $p_1 \ldots p_n$. Note that it is *free* occurrences of 'x' which are ruled out. If p_2 = '(∀x)Fx', say, in which 'x' occurs bound, that does not prevent application of ∀I.

- The rule of ∀E has a new restriction added to it: if you are going to use an individual variable rather than an individual constant in applying ∀E to ⌜$(\forall \nu)\phi\nu$⌝, then the variable which replaces ν must be one which cannot be captured accidentally by a quantifier in $\phi\nu$. The point here is to block the inference from, say, '(∀x)(∃y)Rxy' to '(∃y)Ryy', using 'y' to replace 'x' in an application of ∀E (it is easy to show that (∀x)(∃y)Rxy ⊭ (∃y)Ryy). The phenomenon this mistaken inference illustrates is called *accidental capture*, since the substituted 'y' becomes bound by the existential quantifier, which is not what we intend. The restriction on ∀E which prevents accidental capture is that the variable ν' which is to replace ν

must be such that v has no occurrences in ϕv which are already bound there.

- Like ∀I, in ∃E the same restrictions apply to free occurrences of a variable v as apply to individual constants.

- The rule of ∃I is unchanged; read 'term' for 'individual constant' in the statement of it on p. 186 of *Modern Logic*.

- The rule =E has a new restriction like the one on ∀E: if we replace occurrences of v in ϕv using $v = v'$, then the replacing variable v' must have no bound occurrences in ϕv.