1 Introduction

This tutorial is intended as an introduction to the Nuprl system. It leads you through using Nuprl’s term and proof editor, and gets you to prove a theorem from predicate calculus.

No previous familiarity with the Nuprl system is assumed, but you should be familiar with some presentation of propositional and predicate logic in a two-sided sequent style. By two-sided we mean that introduction rules are given for each connective on each side of the turnstile. The two-sided style is less common than the one-sided style in which introduction and elimination rules are given for each connective on the right hand side of the turnstile. The one-sided style often goes by the name of natural deduction. The two-sided style turns out to be more convenient for automation. In Nuprl, the introduction rules are used bottom-up, so each specifies how to take apart or decompose some connective.

Basic reference material that covers this tutorial can be found in Section 12. For more detailed information on the Nuprl system you should consult the Nuprl Reference Manual and for information about the ML language, the Nuprl ML Manual.

2 Entering Terms in the Term Editor

This tutorial introduces you to using the Nuprl term editor to enter terms. You should first take a quick glance at the chapter of the Nuprl Reference Manual on the ML Top Loop. We illustrate use of the term editor for editing terms by considering the entry of terms from Nuprl’s object language as ML objects of type `term`.

Start with a text cursor at a fresh ML prompt and type `(c-o)`. The ML top loop should look like:

```
ML top loop
M> '[term]' ;;
```

When entering ML text, the command `(c-o)` opens up a position for a term that is to be considered as an ML object of type `term`. The `[term]` is a placeholder. Placeholders are delimited by `[]`’s and include a text string which suggests the kind of term that should be inserted. You get a placeholder when some text or term is missing. The box around `[term]` indicates that a term cursor is positioned at the placeholder. Term cursors are shown on the screen by highlighting.

When you have a term cursor at a placeholder and you type printing characters, the characters are interpreted as the name of a term to insert. Type

```
mul SPACE
```

As the letters `m u l` are typed, they are shown highlighted on the screen. Use `DELETE` to correct any typo’s. After `SPACE`, the system inserts a multiplication term and the ML Top Loop should now look like:

```
ML top loop
M> '[int] * [int]' ;;
```

When you have a term cursor, `SPACE` is a null command and serves to terminate names of terms. The names of some common terms that you might find useful are listed in Section 12.8. Any name not matching a predefined name and including a letter is interpreted as a variable, and a variable term is inserted. If the name entered consists of all numbers, a natural number term is inserted.

Use `MOUSE-LEFT` to move the cursor around. Try clicking `MOUSE-LEFT` on the left `[int]`, the right `[int]` and the `*` symbol. It is easiest to move the cursor around using the mouse, although you can also
use various keyboard commands. If by accident some mouse click results in a new window being opened up, click \texttt{MOUSE-LEFT} in that window and then key \texttt{(c-q)} to close it.

Click \texttt{MOUSE-LEFT} on the left \texttt{[int]} and key 
\[ x \texttt{RETURN} \]
to enter the variable \texttt{x}. You should get the display:

\begin{verbatim}
ML top loop
M> 'x * [int]' ;;
\end{verbatim}

The \texttt{RETURN} key is another useful name terminator. It causes the cursor to jump to the next empty slot in preorder order.

Say that what we really wanted to do was enter the variable \texttt{y} instead of the variable \texttt{x}. To edit the variable name. Click \texttt{MOUSE-LEFT} on the \texttt{x} to get a text cursor to the left of the \texttt{x}, and then key \texttt{(c-d)}. The display should now look like:

\begin{verbatim}
ML top loop
M> '[[variable] * [int]' ;;
\end{verbatim}

The \texttt{[variable]} is an example of a an empty text slot. Enter \texttt{y} to fill it with the character \texttt{y}. The variable term is considered distinct from its text slot. To set a term cursor at a variable term you have to use \texttt{(c-MOUSE-LEFT)}, not \texttt{MOUSE-LEFT}. Try it. \texttt{MOUSE-LEFT} gives you a text cursor if one is appropriate, but \texttt{(c-MOUSE-LEFT)} always gives you a term cursor sitting at the closest surrounding term containing the character you are pointing at.

To delete a term at a term cursor, use \texttt{(c-k)}. Delete the variable \texttt{y}, using \texttt{(c-MOUSE-LEFT)} to first position the cursor at the variable. If you ever mis-type the name of a term and get some other term inserted instead, this is one way of getting rid of it.

Try entering a couple more terms. Starting with the ML Top Loop in state:

\begin{verbatim}
ML top loop
M> '[[int] * [int]' ;;
\end{verbatim}

Enter
\begin{verbatim}
add RETURN 2 RETURN 3 RETURN mul RETURN 4 RETURN 5 RETURN
\end{verbatim}
to get:

\begin{verbatim}
ML top loop
M> '(2 + 3) * 4 * 5' ;;
\end{verbatim}

Note how the editor has an idea about precedence; it automatically inserts parentheses around the \texttt{+}, but not the second \texttt{*}. The cursor stays at the \texttt{5} after the last \texttt{RETURN} because there is no other empty slot to jump to.

The ML evaluator evaluates data objects such objects of type \texttt{term} to themselves. Try evaluating what you’ve just entered. Click \texttt{MOUSE-LEFT} on the space before the \texttt{;;} to get a text cursor, and then key \texttt{RETURN}. In the shell window you should see:
There is a Nuprl ML function `compute` which has type `term -> term`. It invokes an evaluator for terms that lie in Nuprl's object language. To try it, key `c-x` to recall the term you just entered, click `Mouse-Left` on the first `'` character to get a text cursor just before the term, and enter

```
compute space
```

Click `Mouse-Left` on the space between the second `'` and the first `;` to get a text cursor after the Nuprl term. The display should look like:

```
ML top loop
M> compute '(2 + 3) * 4 * 5' ;;
```

Enter `Return` to evaluate and get the result:

```
Shell
M> compute '(2 + 3) * 4 * 5' ;;
100 : term
```

You now should know enough to experiment with typing in other expressions from Nuprl's object language and seeing how they evaluate using the object language evaluator.

### 3 A Sample Theorem in Nuprl’s Logic

We take as an example, the theorem:

$$\forall A: P_i. \forall B: P_i. \neg(A \land B) \iff \neg A \lor \neg B.$$  

This is one of the DeMorgan laws for propositional logic. The connectives $\neg$, $\land$, $\lor$, $\iff$ are respectively the `not`, `and`, `or`, and `iff` connectives of propositional logic. Other logical connectives used in Nuprl are $\Rightarrow$ (`implies`), and $\Leftarrow$ (`is implied by`). We sometimes call $\Leftarrow$ `rev_implies`.

The universal quantifiers (the terms starting with $\forall$) and the $P_i$ in the example perhaps require a little explanation. The expression $\forall x: T. \ P$ reads as “for all $x$ of type $T$, proposition $P$ holds”. The Nuprl logic also has an existential quantifier; $\exists x: T. \ P$ reads as “there exists an $x$ of type $T$ such that proposition $P$ holds”.

It is common in logic books, when there is only one domain over which quantification ranges, to omit any specific mention of the domain. In Nuprl there are many domains or `types` over which quantifiers can range, so we always make the domain explicit. In the theorem above, the domain of quantification for both $A$ and $B$ is $P_i$, read as “propositional universe i” or for short “prop i”. $P_i$ is a type of propositions. The need for the subscript $i$ arises from a concern expounded by the philosopher Bertrand Russell in his *Principia Mathematica*. The concern is that of avoiding circularity of reference. Russell disliked considering the proposition $\forall A: P. \ P$, where $P$ is a type of all propositions, as a member of type $P$. He argued that it is meaningless to `define` some new element of a type by quantifying over all members of that type, since the
range of the quantification would have to already include that new element. He referred to this self-reference as a **vicious circle**.

The Nuprl logic follows in the *type theoretic* tradition established by Russell, and has an infinite series $P_1, P_2, P_3, \ldots$ of types of propositions. The subscripts are called *levels*. $P_2$ is the type of all propositions at level 2. The *level* of some proposition is 1 greater than the maximum of all the levels explicitly mentioned in the proposition. If no levels are mentioned, then the proposition is at level 1. The types $P_i$ are *cumulative* in the sense that $P_i$ contains not just the propositions with level $i$, but the propositions with levels from 1 up to $i$. In Nuprl, we avoid the question of what level to state a theorem at by replacing level numbers by variables, and consider the level variables to be implicitly (universally) quantified over. Hence the use of $P_i$ as the type of $A$ and $B$ in the example theorem.

## 4 Theories

Closely related theorems, definitions, and comments in Nuprl are grouped into *theories*. Theories occupy contiguous sections in the library, and are delimited by easily-identified comment objects. Usually only a subset of the existing theories are loaded in any one Nuprl session.

You should keep theorems and definitions which you develop in your own theory. We describe how to create a theory, how to save it to a file, how load it into a session, and how to print it. All the functions calls listed in this section should be typed in to the ML Top Loop.

### 4.1 Creating a new theory

We describe how to create your own theory.

Scroll the library window to the bottom of the library, and if you haven’t already done so, add in delimiters for your own theory. Enter:

```
bottom () RETURN
add_theory_delimiters "user" RETURN
```

After the `add_theory_delimiters` function is evaluated, theory delimiters appear at the new bottom objects in the library window:

<table>
<thead>
<tr>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>*C class_prop_begin</td>
</tr>
<tr>
<td>*t dneg_elim</td>
</tr>
<tr>
<td>*t imp_elim</td>
</tr>
<tr>
<td>*t neg_imp_elim</td>
</tr>
<tr>
<td>*t neg_or_elim</td>
</tr>
<tr>
<td>*t neg_and_elim</td>
</tr>
<tr>
<td>*t pierce</td>
</tr>
<tr>
<td>*C class_prop_end</td>
</tr>
<tr>
<td>*C user_begin</td>
</tr>
<tr>
<td>*C user_end</td>
</tr>
</tbody>
</table>

### 4.2 Setting the Theory Filename

You should associate your theory with some file-name. This file-name is used whenever saving or loading the theory. Enter:

```
set_theory_filename "user" "(filename)";; RETURN
```
Here, \( \langle \text{filename} \rangle \) should be the name of the file you want to associate with the theory \textit{without} a name extension. Include a pathname. For example, \texttt{~/nuprl/user} would be a valid name. The filename that Nuprl uses for the theory is \( \langle \text{filename} \rangle . \text{thy} \).

### 4.3 Dumping a theory

To save your theory to a file, enter at the ML Top Loop

\[
dump \text{_theory} "\text{user}" \text{RETURN}
\]

This only works if the theory is currently loaded into Nuprl's library and you have previously set the theory's file-name. Setting the theory filename is explained in Section 4.2.

### 4.4 Loading an Existing Theory

If you have started a new Nuprl session, and you want to load your previously saved theory, enter:

\[
\text{load}_{\_\text{theory}} "\text{user}" \text{RETURN}
\]

The load only succeeds if the theory filename has been set as explained in Section 4.2, and there isn't a theory by the same name already loaded. The theory is always loaded at the bottom of the library.

### 4.5 Printing a Theory

Print a loaded theory using:

\[
\text{print}_{\_\text{theory}} "\text{user}" \text{RETURN}
\]

If \( \langle \text{file-name} \rangle \) is the file-name associated with your \texttt{user} theory, then two files are generated; a text file \( \langle \text{file-name} \rangle . \text{prl} \) and a self-contained \LaTeX\ file \( \langle \text{file-name} \rangle . \text{tex} \).

### 5 Creating a Theorem

Create a theorem object. Enter:

\[
\text{create}_{\_\text{thm}} "\text{not\_over\_and}" "\text{user\_end}" \text{RETURN}
\]

The library window should now look like:

```
Library
*\text{C class\_prop\_begin}  ************ CLASS PROP ***************
*\text{t dneg\_elim}  \forall A: \text{P\{i\}}. \neg \neg A \Rightarrow A
*\text{t imp\_elim}  \forall A: \text{P\{i\}}. A \Rightarrow B \Rightarrow \neg A \lor B
*\text{t neg\_imp\_elim}  \forall A: \text{P\{i\}}. \forall B: \text{P\{i\}}. \neg (A \Rightarrow B) \Rightarrow A \land \neg B
*\text{t neg\_or\_elim}  \forall A: \text{P\{i\}}. \forall B: \text{P\{i\}}. \neg (A \lor B) \Rightarrow \neg A \land \neg B
*\text{t neg\_and\_elim}  \forall A: \text{P\{i\}}. \forall B: \text{P\{i\}}. \neg (A \land B) \Rightarrow \neg A \lor \neg B
*\text{t pierce}  \forall P: \text{P\{i\}}. \forall Q: \text{P\{i\}}. ((P \Rightarrow Q) \Rightarrow P) \Rightarrow P
*\text{C class\_prop\_end}  ************ USER ****************
*\text{C user\_begin}  ******************* USER **********************
*\text{t not\_over\_and}  [\text{term}]
*\text{C user\_end}  ******************* USER **********************
```

View the theorem. Enter:

\[
\text{view} "\text{not\_over\_and}" \text{RETURN}
\]
Nuprl opens up a *proof editor* window. The window should appear as:

```
EDIT THM not_over_and
? top
<main proof goal>
```

Move the mouse cursor over into the new window, and click **MOUSE-LEFT**. The cursor in the ML-Top-Loopwindow vanishes, and a new cursor appears in the proof window. In general, **MOUSE-LEFT** can be used to switch to different Nuprl windows. Only one window at a time has a cursor. The window with the cursor receives the input from keyboard strokes and mouse clicks. Often but not always, this window is the same as the window containing the mouse; there are a couple of keyboard commands which move the cursor between windows without moving the mouse.

To Exit a proof window, use `I c-z`. This saves the proof and closes the window.

### 6 Entering the Main Goal of a Theorem

Click **MOUSE-RIGHT** on `<main proof goal>`. Nuprl opens up a *term editor* window. The window initially looks like:

```
EDIT main goal of not_over_and

[term]
```

The main goal is entered in a structural top-down fashion. Enter:

```
all \text{RETURN}
```

After the `\text{RETURN}`, the display should look like:

```
EDIT main goal of not_over_and

\forall [var]:[type]. \text{[prop]}
```

The `[var]` slot of the `all` term is a text slot. The `[type]`, and `[prop]` slots are term slots. Try clicking **MOUSE-LEFT** at each of of the slots. Notice how you get a text cursor for a text slot and a term cursor for the term slots.

Click **MOUSE-LEFT** on `var` and enter:

```
A \text{RETURN}
```

The character `\text{A}` gets inserted in the variable slot, and on keying `\text{RETURN}`, the cursor moves onto the `[type]` slot. Enter:

```
\text{prop} \text{RETURN} i \text{RETURN}
```

The term `[level]` gets inserted, and a text cursor is positioned at the start of `[level]`. The display should now look like:
The \textit{prop} you just entered has nothing to do with the \textit{prop} in the \textit{[prop]} place-holder. \textit{prop} happens to be a name for the \textit{propositional universe} term.

In a similar way, you should now be able to enter a second universal quantifier to give the following situation:

\begin{verbatim}
EDIT main goal of not_over_and
\forall A: \mathbb{P}\{i\}. \forall B: \mathbb{P}\{i\}. \text{ [prop]}
\end{verbatim}

Now enter:

\texttt{iff [RETURN] not [RETURN] or [RETURN]}

The display should now look like:

\begin{verbatim}
EDIT main goal of not_over_and
\forall A: \mathbb{P}\{i\}. \forall B: \mathbb{P}\{i\}. \neg(\text{[prop]} \land \text{[prop]}) \iff \text{[prop]}
\end{verbatim}

Note that Nuprl automatically inserted parentheses around the \textit{[prop]} \land \textit{[prop]} term. We have given Nuprl a \textit{precedence} for each term that might need parenthesizing. Nuprl uses this precedence information to always automatically insert parentheses as appropriate. Related to precedence, we can optionally give binary infix terms an \textit{associativity} of either ‘L’ or ‘R’. ‘L’ means that parentheses should not be used when the term associates to the left. Similarly, ‘R’ suppresses the use of parentheses when the term associates to the right. For example, \texttt{\langle \land \rangle} associates to the left, so Nuprl displays \texttt{(A \land B) \land C} as \texttt{A \land B \land C}

but \texttt{\langle A \land B \rangle} as:

\texttt{A \land (B \land C)}

To continue, with the cursor at the left \textit{[prop]} of \textit{[prop]} \land \textit{[prop]}, enter:

\texttt{A [RETURN]}

to insert the variable term \textit{A}.

You should now know enough to complete entering the main goal, and correct any mistakes you make. The completed main goal should look like:

\begin{verbatim}
EDIT main goal of not_over_and
\forall A: \mathbb{P}\{i\}. \forall B: \mathbb{P}\{i\}. \neg(\text{[prop]} \land \text{[prop]}) \iff \neg A \lor \neg B
\end{verbatim}

If a term won’t fit on one line, Nuprl inserts linebreaks and indentation as appropriate. Sometimes a window can be made easier to read by resizing it. Nuprl automatically adjusts the formatting when the window is resized. Occasionally, some subterm is \textit{elided}, and shown as

\texttt{( . . . )}

In this event, the elided subterm can often be made visible by widening the window.

Once you have finished entering the main goal, close the main goal window by keying \texttt{<c-z>}. 
7 Running Tactics 1

The proof-editor window should now look like:

```
EDIT THM not_over_and
# top
\forall A: P\{i\}. \forall B: P\{i\}. \neg (A \land B) \iff \neg A \lor \neg B
```

Click **MOUSE-RIGHT** on `<refinement rule>`. **MOUSE-RIGHT** selects it and opens up a term-editor window, which should look like:

```
EDIT rule of not_over_and
```

The rule window is set up for entering a tactic. Initially the window contains an empty text string. Enter the tactic:

```
D 0 (c-z)
```

Here `0` is the number zero. The letter O in the typewriter font is `0`. The `(c-z)` closes the window and runs the tactic `D 0` on the main goal, generating two subgoals:

```
EDIT THM not_over_and
# top
\forall A: P\{i\}. \forall B: P\{i\}. \neg (A \land B) \iff \neg A \lor \neg B

BY D 0

1# 1. A: P\{i\}
\verbatim
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
\verbatim:
```

The D tactic decomposes the outermost connective of either a hypothesis or the conclusion. The D tactic always takes an integer argument. To see this, click **MOUSE-LEFT** in the ML-Top-Loop window, and enter

```
D RETURN
```

You should see:

```
M> D;;
- : int -> tactic
M> |
```
So the expression \( D \ 0 \) has the ML type `tactic`. Enter

\[
D \ 0 \ \text{return}
\]

You should see:

```
M> D 0;;
- : tactic
M> []
```

## 8 Well-Formedness of Terms

This section explains why Nuprl’s basic inference rules have *well-formedness* premisses (or subgoals, thinking of the way in which Nuprl uses the rules), and motivates the use of the `TypeCheck` tactic.

Nuprl takes an uncommon attitude towards the checking of the *well-formedness* of terms. A term is well-formed if it has been properly put together. Usually in the study of logic, well-formedness is characterized by a simple set of syntactic rules, and when building theorem provers, it is easy to check well-formedness automatically. Because Nuprl is intended for for general purpose mathematics, things are not quite as simple. For example consider the proposition

\[
\forall j, k: \mathbb{N}. \ k \neq 0 \Rightarrow j \div k \leq j
\]

Here, it is not true that \( j \div k \) is a meaningful expression for all \( j \) and \( k \) of type \( \mathbb{N} \) \((\{0, 1, 2, 3, \ldots\}\); it is meaningless when \( k = 0 \). So to check the proposition’s well-formedness properly, we need to take note of the \( k \neq 0 \) on the left-hand side of the implication. In general \( k \) might be replaced by some complicated expression, in which case it might take a fair amount of work to prove that the proposition is well-formed. For this reason, Nuprl’s logical rules were designed such that well-formedness checking is done simultaneously with truth checking. The rules guarantee that if a proof of a theorem is completed, then the theorem is both well-formed and true.

The second subgoal, with \( P\{i\} \in U\{i’\} \) as the conclusion, is a well-formedness subgoal, requesting that the well-formedness of \( P\{i\} \) be checked. Well-formedness checking is always phrased as type checking; Some expression is well formed if it is a member of some appropriate type. Types themselves are considered members of *type universes* or just *universes* for short. As with propositional universes discussed earlier, to avoid a vicious circle, we have an infinite series \( U_1, U_2, U_3, \ldots \) of type universes, where the subscripts are levels. Similar rules apply for determining the level of any term; take the maximum of the levels occurring in the term and add 1. If no levels occur, then the level is 1. \( U_i \) contains as elements all types with levels from 1 to \( i \). The prime on the \( i \) in \( U\{i’\} \) is shorthand for ‘successor’, so \( U\{i’\} \) is the same as \( U\{i+1\} \).

## 9 Running Tactics 2

Type-checking subgoals for basic propositional and predicate logic can always be dealt with automatically, and we have a tactic `Auto` for that purpose.

Return to the proof editor window using `Mouse-Left`, and click `Mouse-Right` on the tactic `D 0` which you just typed in. A term-editor window opens up again. Click `Mouse-Left` after the 0 and enter:

```
RETURN THENW Auto
```

The rule window should look like:
To run the tactic, enter \( \texttt{(c-z)} \). The rule window closes and once the tactic has executed, the proof window should look like:

EDIT THM not_over_and

\[
\begin{align*}
\# & \text{ top} \\
\vdash & \forall A: \mathbb{P}\{i\}. \forall B: \mathbb{P}\{i\}. \neg (A \land B) \iff \neg A \lor \neg B \\
\end{align*}
\]

\[
\begin{align*}
\text{D 0} \\
\text{THENW Auto} \\
\end{align*}
\]

\[
\begin{align*}
\# & \text{ 1. A: \mathbb{P}\{i\}} \\
\vdash & \forall B: \mathbb{P}\{i\}. \neg (A \land B) \iff \neg A \lor \neg B \\
\end{align*}
\]

THENW is a tactical, a combinator for composing tactics. It has been declared to be an ML infix function, so one argument comes before and one after. Nearly all ML infix functions have names of more than one character, with all upper-case characters. To look at its type, enter

\[
\texttt{\$THENW \textsc{RETURN}}
\]

in the ML Top Loop. $\texttt{\$THENW}$ is a version of \texttt{THENW} which behaves like other ML functions in that all its arguments are placed to its right. What gets printed out should be:

\[
\begin{align*}
\text{M> } & \texttt{\$THENW};; \\
\rightarrow & : \text{ tactic } \rightarrow \text{ tactic } \rightarrow \text{ tactic} \\
\text{M> } & \texttt{} \\
\end{align*}
\]

THENW is a sequencing tactical; The tactic $T_1 \texttt{\ THENW } T_2$ first runs $T_1$ on some goal, and then runs $T_2$ on any well-formedness subgoals generated by $T_2$. When doing Nuprl proofs you should add \texttt{THENW Auto} after any other tactics which generate well-formedness subgoals. Well-formedness subgoals are pretty easy to recognize; they always have a conclusion of form \( \neg e \in \mathbb{T} \) where \( e \) is some expression, and \( \mathbb{T} \) is a type. The type universes \( \mathbb{U}\{i\} \) are themselves considered types. Other examples of types which you will encounter later are \( \mathbb{Z} \), the integers, and \( \mathbb{B} \), the booleans.

The first time you run a new tactic, or an existing tactic which might act in a new way, its worth running it without the \texttt{THENW Auto} just to see how it works. Thereafter, always use the \texttt{THENW Auto}. From here on in the tutorial, we always include the \texttt{THENW Auto} if it is necessary, but feel free to try leaving it out first.

\section{Walking About a Proof}

You can move up and down a proof tree using the \texttt{\textsc{Mouse-Middle}}. Clicking \texttt{\textsc{Mouse-Middle}} in some subgoal of the proof window, makes that subgoal the goal of the window. Once you are below the top level of a proof, clicking \texttt{\textsc{Mouse-Middle}} in the goal of the window moves the proof window up to the parent goal. Click \texttt{\textsc{Mouse-Middle}} in the only subgoal generated by \texttt{D 0 \ THENW Auto}. The proof window should look like
Now Click **MOUSE-MIDDLE** in the goal of this window. The proof window moves back to where it was before. Feel free during the rest of this tutorial to go wandering around the proof tree using **MOUSE-MIDDLE**. Just make a note first of where you are, so you can find your way back and continue the tutorial. The top line of the proof window always gives the tree address of the node the proof window is at.

### 11 Running Tactics 3

Execute the D 0 THENW Auto tactic by entering (c-z) and move down into the new subgoal. The proof window should look like:

Now to understand how $\iff$ decomposes, look at its definition:

$$P \iff Q =_{\text{def}} (P \Rightarrow Q) \land (P \Leftarrow Q)$$

So, $\iff$ decomposes the same way that $\land$ decomposes. Try D 0 on it. (There is no need for THENW Auto since the decomposition rule for $\land$ in the conclusion, doesn’t generate any well-formedness subgoals. After running D 0, the proof window should look like:

Move down into each of these subgoals in turn, and run D 0 THENW Auto. The two proof nodes should look like
The partial proof you have generated so far is rather gangly and repetitive. All the steps involved running the same tactic (D 0) on the conclusion. Go back to the top of the proof and run instead the tactic GenUnivCD THENW Auto to get:

Isn’t that better! GenUnivCD repeats D 0 on all, implies, is implied by, and iff terms. Sometimes GenUniv does a bit much, in which case UnivCD is the tactic to use. UnivCD is like GenUnivCD but it stops at and and iff terms.

Move down into the first subgoal and open the rule box. At this point in the proof, it is appropriate to do a case split on whether A or ¬A is true. That is, you need to invoke the classical rule of the excluded
middle. The tactic to use is \texttt{ClassDecide}. It has ML type $\texttt{term} \rightarrow \texttt{tactic}$; it takes as its argument the proposition to do the case split on. Enter:

\begin{verbatim}
ClassDecide
\end{verbatim}

Now enter \texttt{(c-o)}. The window should look like

<table>
<thead>
<tr>
<th>EDIT rule of \texttt{not} _ \texttt{over} _ \texttt{and}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ClassDecide ' [term]'}</td>
</tr>
</tbody>
</table>

Here we are supplying a term from Nuprl’s object language as an argument to an ML function, just as we did in Section 2. Enter \texttt{A} \texttt{RETURN} \texttt{(c-z)} and the tactic executes, giving the proof window:

<table>
<thead>
<tr>
<th>EDIT THM \texttt{not} _ \texttt{over} _ \texttt{and}</th>
</tr>
</thead>
<tbody>
<tr>
<td># top 1</td>
</tr>
<tr>
<td>1. $A$: $P{i}$</td>
</tr>
<tr>
<td>2. $B$: $P{i}$</td>
</tr>
<tr>
<td>3. $\neg (A \land B)$</td>
</tr>
<tr>
<td>$\vdash \neg A \lor \neg B$</td>
</tr>
</tbody>
</table>

\begin{verbatim}
BY ClassDecide 'A'
\end{verbatim}

$1\# 4. \ A \vdash \neg A \lor \neg B$

$2\# 4. \ \neg A \vdash \neg A \lor \neg B$

The 2nd subgoal is easily proved by applying the tactic \texttt{Sel 1 (D 0) THENW Auto}, and then \texttt{Hypothesis}. Try first by running each of these tactics at separate proof nodes. Then, use the \texttt{THENM} tactical and run instead \texttt{(Sel 1 (D 0) THENM Hypothesis) THENW Auto}. $T_1$ \texttt{THENM} $T_2$ runs $T_2$ only on the main subgoal(s) produced by $T_1$. Well-formedness subgoals are not considered to be main subgoals. Infix tacticals in ML all have the same parsing precedence, and associate to the left. So, the parentheses around \texttt{Sel 1 (D 0) THENM Hypothesis} are not strictly necessary. Still, they improve readability.

It is possible to compress the depth of a proof significantly, by use of tacticals such as \texttt{THENM}$. It's instructive to each in turn!

You now know enough that you should be able to complete the proof of this theorem, and go onto prove similar theorems from propositional calculus. You almost have enough for proving theorems from predicate calculus. The main new things you need to know are how to express the theorems in Nuprl’s logic, and how both quantifiers decompose on either side of the turnstile (the $\vdash$ symbol in sequents which separates the hypotheses on the left from the conclusion on the right).

Note that the theorem we proved is a theorem of classical, not constructive logic. Constructive logic does not permit the unrestricted use of case splits on whether some proposition or its negation is true. (Hence the name of the tactic \texttt{ClassDecide}.)

\footnote{Nuprl users have been known to reduce their entire proof to one monolithic tactic!}
12 Reference Material for Tutorial

12.1 Term Editor

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOUSE-LEFT</td>
<td>Jump to text / term at mouse cursor. Switch windows</td>
</tr>
<tr>
<td>(c- MOUSE-LEFT)</td>
<td>Jump to term at mouse cursor. Switch windows</td>
</tr>
<tr>
<td>RETURN</td>
<td>1. In ML text of Top Loop: Evaluate expression.</td>
</tr>
<tr>
<td></td>
<td>2. In ML text elsewhere: Insert newline</td>
</tr>
<tr>
<td></td>
<td>3. Otherwise: Jump to empty slot</td>
</tr>
<tr>
<td>(c- RETURN)</td>
<td>In ML text of Top Loop: Insert newline</td>
</tr>
<tr>
<td>(c-k)</td>
<td>With term cursor: delete (and save) term</td>
</tr>
<tr>
<td>(c-d)</td>
<td>With text cursor: delete char to right</td>
</tr>
<tr>
<td>DELETE</td>
<td>With text cursor: delete char to left</td>
</tr>
<tr>
<td>(c-o)</td>
<td>In ML text: open position for term</td>
</tr>
<tr>
<td>(c-q)</td>
<td>Quit. Don’t act on, or save any changes</td>
</tr>
<tr>
<td>(c-z)</td>
<td>Exit. Save contents if appropriate.</td>
</tr>
<tr>
<td></td>
<td>If tactic in window, run it</td>
</tr>
<tr>
<td>(c-r)</td>
<td>In Top Loop: scroll back in history</td>
</tr>
<tr>
<td>(M-r)</td>
<td>In Top Loop: scroll forward in history</td>
</tr>
</tbody>
</table>

Table 1: Basic Commands for Term Editor Windows

Table 1 summarizes commands for term editor windows.

12.2 Proof Editor

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOUSE-LEFT</td>
<td>Jump to this window</td>
</tr>
<tr>
<td>MOUSE-MIDDLE</td>
<td>Move proof window into indicated goal / subgoal</td>
</tr>
<tr>
<td>MOUSE-RIGHT</td>
<td>Select main goal, rule or subgoal for viewing / editing</td>
</tr>
<tr>
<td>(c-t)</td>
<td>Open transformation tactic window.</td>
</tr>
<tr>
<td>(c-z)</td>
<td>Exit Proof.</td>
</tr>
<tr>
<td>(c-f)</td>
<td>Move down within window</td>
</tr>
<tr>
<td>(c-b)</td>
<td>Move up within window</td>
</tr>
</tbody>
</table>

Table 2: Commands for Proof Editor

Proof editor commands are summarized in Table 2.

12.3 ML Functions for ML Top Loop

The most important ML-Top-Loop functions are summarized in Table 3. Notes on functions:

- *name, place, thy* and *fname* should be enclosed in `"`’s.
- *name* and *thy* should all use only alphanumeric characters and underscore (`_`). *place* should start with `+` or `-` to indicate position immediately before or after the object named.
- *fname* should include a full pathname but no extension. Files:
  - *fname.prl* text file with special characters.
  - *fname.tex* \LaTeX file for input to latex.
  - *fname.dvi* Output file from \LaTeX.
  - *fname.thy* theory file. (Lisp Expressions).
Table 3: Functions for ML Top Loop

12.4 Tactics

<table>
<thead>
<tr>
<th>D i</th>
<th>Decompose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.hyps ((i &gt; 0)): and, or, implies, not, exists</td>
<td></td>
</tr>
<tr>
<td>2.concl ((i = 0)): and, implies, all, not</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID i</th>
<th>Decompose implies, not, all in hyp without thinning</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Sel n (D 0)</th>
<th>Decompose or in concl. ((n = 1, 2))</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>With t (D i)</th>
<th>Instantiate with term (t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. all in hyp 2. exists in concl</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UnivCD</th>
<th>D 0 on implies, all</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenUnivCD</td>
<td>D 0 on implies, all, and</td>
</tr>
<tr>
<td>ClassDecide P</td>
<td>Case split on whether (P) or (\neg P) is true</td>
</tr>
<tr>
<td>Trivial</td>
<td>1. true in concl.</td>
</tr>
<tr>
<td></td>
<td>2. false in hyp.</td>
</tr>
<tr>
<td></td>
<td>3. (P) and (\neg P) in hyps.</td>
</tr>
<tr>
<td></td>
<td>4. (P) in concl and some hyp.</td>
</tr>
<tr>
<td>Auto</td>
<td>Solve goals of form (t \in T).</td>
</tr>
<tr>
<td>(T_1) THENM (T_2)</td>
<td>Run (T_1), then (T_2) on main subgoals</td>
</tr>
<tr>
<td>(T_1) THENW (T_2)</td>
<td>Run (T_1), then (T_2) on wf subgoals</td>
</tr>
</tbody>
</table>

Table 4: Tactics

Some useful tactics are summarized in Table 4.

12.5 Printing Transcripts

To print up a transcript file, you can use the ML function `latexize_file` to create a \LaTeX\Xable version of it. `latexize_file` has ML type `string -> string -> unit`. The first argument should be the name of a transcript file. The second the name of the file you want the \LaTeX\Xized version of the transcript written to. For example, if you have written a transcript to the file `~/tx1.prl`, you might type:

ML top loop

```ml
M> latexize_file "~/tx1.prl" "~/tx1.tex" ;;
```
12.6 Printing a Proof Tree

The proof editor only allows you to see a small section of a proof at a time, but you can also create a \texttt{LaTeX} file which contain printed representation of a whole proof tree or sub-tree.

First move a proof editor window such that the goal in the proof window is either at the root of the whole proof or at some sub-tree you want to print. Then enter \texttt{(c-t)}. A term-editor window called a \textit{transformation tactic} window opens up, which should look like:

<table>
<thead>
<tr>
<th>Transformation Tactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{term} ]</td>
</tr>
</tbody>
</table>

Transformation tactics are tactics which transform a whole proof tree. The tactics you have encountered so far are all \textit{refinement tactics} which only work on the leaves of a proof tree. The transformation tactic you run to print a proof works by side-effect; it doesn’t change the proof in any way. Enter:

\texttt{PrintTexFile "(file-name)"}

Here, \texttt{(file-name)} should include a full pathname, and should not include an extension.

Enter \texttt{(c-z)} to run the print tactic and exit the transformation tactic window. Two files are created: the \texttt{LaTeX} file \texttt{(file-name).tex} and the file \texttt{(file-name).prl} which can be viewed using emacs running with a nuprl font.

12.7 Running \texttt{LaTeX}

Useful \texttt{LaTeX}-related unix shell commands are:

1. \texttt{latex (file-name)}
   to run \texttt{LaTeX} on the \texttt{.tex} file and generate a \texttt{(file-name).dvi} file.

2. \texttt{xdvi (file-name)}
   to view the \texttt{.dvi} file on your screen.

3. \texttt{dvips (file-name) | lpr -P(printer-name)}
   to print the \texttt{.dvi} file.

12.8 Nuprl Terms

Table 5 lists terms in the computational fragment of Nuprl’s object language. Table 6 lists the terms for types and Table 7 lists the terms for propositional and predicate logic.

The ‘Prec’ column in the tables is for the precedence of the term. The ‘Assoc’ column gives the associativity of infix terms. This information simplifies slightly how the pretty-printer calculates when to add parentheses. In practice, slightly more parentheses might be inserted than you might expect from the information in the tables.
<table>
<thead>
<tr>
<th>Term</th>
<th>Name</th>
<th>Prec</th>
<th>Assoc</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[var]</td>
<td>var</td>
<td></td>
<td></td>
<td>variable</td>
</tr>
<tr>
<td>(\lambda [var].[term])</td>
<td>lam</td>
<td>6</td>
<td>L</td>
<td>lambda</td>
</tr>
<tr>
<td>[term] [term]</td>
<td>ap</td>
<td>10</td>
<td>L</td>
<td>apply</td>
</tr>
<tr>
<td>&lt;[term],[term]&gt;</td>
<td>pair</td>
<td></td>
<td></td>
<td>pair</td>
</tr>
<tr>
<td>[pair term].1</td>
<td>pi1</td>
<td>11</td>
<td>L</td>
<td>lhs of pair</td>
</tr>
<tr>
<td>[pair term].2</td>
<td>pi2</td>
<td>11</td>
<td>L</td>
<td>rhs of pair</td>
</tr>
<tr>
<td>[natural]</td>
<td>natnum</td>
<td></td>
<td></td>
<td>natural number</td>
</tr>
<tr>
<td>-[int]</td>
<td>minus</td>
<td>10</td>
<td></td>
<td>unary minus</td>
</tr>
<tr>
<td>[int] * [int]</td>
<td>mul</td>
<td>9</td>
<td>L</td>
<td>multiply</td>
</tr>
<tr>
<td>[int] + [int]</td>
<td>add</td>
<td>8</td>
<td>L</td>
<td>add</td>
</tr>
<tr>
<td>[int] - [int]</td>
<td>sub</td>
<td>8</td>
<td></td>
<td>subtract</td>
</tr>
</tbody>
</table>

Table 5: Terms in Nuprl's computation language

<table>
<thead>
<tr>
<th>Term</th>
<th>Name</th>
<th>Prec</th>
<th>Assoc</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathbb{P}{[level]})</td>
<td>prop</td>
<td></td>
<td></td>
<td>propositional universe type universe</td>
</tr>
<tr>
<td>(\mathbb{U}[level])</td>
<td>univ</td>
<td></td>
<td></td>
<td>type universe</td>
</tr>
<tr>
<td>[prop] \times [prop]</td>
<td>prod</td>
<td>6</td>
<td>R</td>
<td>product type</td>
</tr>
<tr>
<td>[prop] \rightarrow [prop]</td>
<td>fun</td>
<td>6</td>
<td>R</td>
<td>function type</td>
</tr>
<tr>
<td>(\mathbb{Z})</td>
<td>int</td>
<td></td>
<td></td>
<td>integer type</td>
</tr>
</tbody>
</table>

Table 6: Types

<table>
<thead>
<tr>
<th>Term</th>
<th>Name</th>
<th>Prec</th>
<th>Assoc</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>true</td>
<td></td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>False</td>
<td>false</td>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>[int] \leq [int]</td>
<td>le</td>
<td>5</td>
<td>R</td>
<td>less than or equal</td>
</tr>
<tr>
<td>[int] &lt; [int]</td>
<td>lt</td>
<td>5</td>
<td>R</td>
<td>less than</td>
</tr>
<tr>
<td>[term] = [term] \in [type]</td>
<td>equal</td>
<td>5</td>
<td>R</td>
<td>equality</td>
</tr>
<tr>
<td>[int] = [int]</td>
<td>eqi</td>
<td>5</td>
<td>R</td>
<td>equality over (\mathbb{Z}) (abbr. of equal)</td>
</tr>
<tr>
<td>\neg[prop]</td>
<td>not</td>
<td>6</td>
<td></td>
<td>not</td>
</tr>
<tr>
<td>[prop] \land [prop]</td>
<td>and</td>
<td>4</td>
<td>L</td>
<td>and</td>
</tr>
<tr>
<td>[prop] \lor [prop]</td>
<td>or</td>
<td>4</td>
<td>L</td>
<td>or</td>
</tr>
<tr>
<td>[prop] \Rightarrow [prop]</td>
<td>implies</td>
<td>3</td>
<td>R</td>
<td>implies</td>
</tr>
<tr>
<td>[prop] \iff [prop]</td>
<td>rev_implies</td>
<td>3</td>
<td>L</td>
<td>is implied by</td>
</tr>
<tr>
<td>[prop] \iff [prop]</td>
<td>iff</td>
<td>2</td>
<td></td>
<td>if and only if</td>
</tr>
<tr>
<td>\forall [var].[type]. [prop]</td>
<td>all</td>
<td>1</td>
<td></td>
<td>universal quantifier</td>
</tr>
<tr>
<td>\exists [var].[type]. [prop]</td>
<td>exists</td>
<td>1</td>
<td></td>
<td>existential quantifier</td>
</tr>
</tbody>
</table>

Table 7: Propositions