TkWinHOL
A Tool for Doing Window Inference in HOL

Thomas Långbacka   Rimvydas Rukšėnas   Joakim von Wright

Åbo Akademi University
Department of Computer Science
Lemminkäinengatan 14–18
SF–20520 Åbo, Finland

April 28, 1995

Abstract

Window inference is a method for contextual rewriting and refinement, supported by the HOL Window Inference Library. This paper describes a user-friendly interface for window inference. The interface permits the user to select subexpressions by pointing and clicking and to select transformations from menus. The correctness of each transformation step is proved automatically by the HOL system. The interface can be tailored to particular user-defined theories. One such extension, for program refinement, is described.

1 Introduction

Though the original purpose of the HOL system [8] was as a tool for hardware verification, it has become popular also as a basis for software verification (see for example [1, 5, 7]). However, the theories built for supporting the software development process are normally difficult to use, especially if one does not have any previous detailed knowledge of the HOL system. In order to make such theories available to a general audience, it is essential that users have support in the form of a user interface, either a general HOL interface as [19] or a specialised tools as [2]. This observation has motivated us to design a tool that supports development of provably correct programs within the framework of the Refinement Calculus [3, 4, 6].

Our work is based on a number of layers. The starting point is the HOL system and its Window Inference library [9, 10]. Window inference (originally described in [17]) supports the notion of contextual refinement which is an important ingredient in the refinement calculus as well. We also use the HOL theory for the Refinement Calculus [20]. On top of these packages we have designed a graphical user interface that itself is designed in a layered fashion. The basic user interface is called TkWinHOL, and it offers users a X-windows based graphical interface to the window inference system.

TkWinHOL is built using the Tcl/Tk programming system [15], which is a general purpose environment for building user interfaces. We have managed to keep the system small but still efficient. TkWinHOL is also easy to extend and modify. Based on the TkWinHOL system we are developing a tool for the Refinement Calculus.

2 Window inference

In window reasoning [17], the user transforms a term so that a *reflexive transitive relation* R is preserved.

The user can also restrict attention to a subterm of the term that is being transformed. A transformation of the subterm is at the same time a transformation of the whole term, provided that certain context-dependent monotonicity conditions hold.

In the HOL *Window Inference library* [9, 10], there is built-in support for window reasoning under equality, implication and reverse implication and the user can add other relations. In HOL, window reasoning is handled as follows. Assume that $E[\ldots t \ldots]$ is a term with subterm t. Our starting point is the *window*

$$R \;\star\; E[\ldots t \ldots]$$

The *focus* of the window is the term $E[\ldots t \ldots]$ and the relation that we want to preserve (the *window relation*)is R (the star simply separates these two). When the HOL Window Inference library has been loaded into the HOL system, a *window stack* with this window is set up by typing the appropriate command.

The theorem associated with this window (the *window theorem*) is

$$\vdash \; E[\ldots t \ldots] \; R \; E[\ldots t \ldots]$$

which holds because R is reflexive.

Opening subwindows We can *open a subwindow* on the subterm t. This is done by entering a command of the form `DO(OPEN_WIN path)`, where `path` is a list of path elements that indicate how we navigate in the abstract syntax tree of E in order to get to the subterm t. In view of the simple syntax of higher order logic, the possible path elements are `RATOR`, `RAND` and `BODY` (we assume that the reader is familiar with higher order logic and has some experience of the HOL system). The result of the opening command is the following window:

$$R' \;\star\; t$$

with the corresponding window theorem (again by reflexivity)

$$\vdash \; t \; R' \; t$$

Here R' may be the same relation as R, but it can be different. For example, R can be implication and R' equality (this would be the case if the focus was $x + 1 - 1 \leq y$, the relation was implication and we opened a subwindow on $x +$

1−1). We now transform t under the relation R', using available transformation rules (e.g., rewriting, beta-conversions, matching with pre-proved theorems etc). Assume that the result of these transformations is t'. The new window theorem is then

$$\vdash\ t\ R'\ t'$$

(the system uses transitivity of R' to draw this conclusion). We can now close the subwindow by entering the command `DO CLOSE_WIN`.

Closing At this point, the window inference system uses a collection of *window closing rules* (ML functions that perform HOL proofs) to do the following inference:

$$\frac{\vdash\ t\ R'\ t'}{\vdash\ E[\ldots t\ldots]\ R\ E[\ldots t'\ldots]}$$

Essentially, closing rules embody homomorphism properties of contexts. The system chooses suitable closing rules automatically (the choice is made when the subwindow is opened); the user does not have to know names or other details of these rules. The result of closing is the new window

$$R \star E[\ldots t'\ldots]$$

and the window theorem shows the result of the the whole sequence of steps:

$$\vdash\ E[\ldots t\ldots]\ R\ E[\ldots t'\ldots]$$

We can now store this theorem in a theory file or we can continue and transform $E[\ldots t'\ldots]$ further.

Assumptions The window inference system keeps tracks of what assumptions can be used in a transformation. Consider opening a subwindow on the right conjunct of the following window:

$$\Rightarrow\ \star\ \mathsf{x} = 1 \wedge \mathsf{x} > 0$$

The new window has $\mathsf{x}{=}1$ as an *assumption*:

$$\begin{array}{rl} ! & \mathsf{x} = 1 \\ \Rightarrow\ \star & \mathsf{x} > 0 \end{array}$$

The assumption can be used to transform the focus to `T` (truth). After this we can close and rewrite; this transforms the original focus to $\mathsf{x}{=}1$.

Adding new relations The Window Inference library lets the user add a new relation R into the system, once R has been proved reflexive and transitive. An example is the refinement relation on programs which is used in the example in Section 5. In addition to adding a relation, the user can also add window closing rules. In the refinement example, we have added one rule for each argument of each statement constructor (e.g., one rule for closing after opening a subwindow on the left component of a sequential composition).

3 TkWinHOL

TkWinHOL is a graphical user interface to the window library. It runs on UNIX based systems under X-windows. It requires the availability of the Tcl/Tk [15] programming system and the expect [14] extension to Tcl/Tk. The HOL environment supported is HOL90.

TkWinHOL is designed to simplify the work a user has to do while transforming an initial focus towards the desired goal. One of the main features of TkWinHOL is that it gives the user the possibility to use the mouse to point out subterms that need to be operated on. The user is spared from the task of typing in explicit path information which can be tedious if operating on a large focus. The system has other interesting features as well (see below) and we have tried to build in as much graphical support as possible.

Using Tcl/Tk for implementing TkWinHOL makes the system small yet powerful. It also makes the system 'open' in the sense that it can easily be modified and extended. Indeed, one of the basic ideas of TkWinHOL is that it should be possible to use as a basic building block for the more specialised system discussed in Section 5. Tcl/Tk has turned out to be a good choice also because of the fact that there is work going on using Tcl/Tk for interfacing to other aspects of the HOL system that we haven't considered (for example theorem retrieval) [18], and there is a great potential for cooperation.

3.1 The basic appearance of TkWinHOL

When TkWinHOL is started three windows appear on the screen. First there is a simple text editor window that offers a small set of menu commands for file and edit operations. The text editor window also offers a subset of the ordinary emacs key sequences for moving the insertion point etc. However, the user does not have to use this editor (this is explained below).

The second window (the HOL window) is similar to a terminal window (like xterm). Users can execute any HOL command they wish by selecting (using the mouse) a text string containing the command and pressing an execute button that is associated with the HOL window. The text string is sent verbatim to the HOL process running in the background and the reply given by HOL is presented in the HOL window. The selected text can be in the text editor window but it can also be in a separate window (that is, a window not opened by the TkWinHOL application) running, for example, emacs (provided that the emacs version used exports the selection).

The third window is the heart of the application. This window (called the focus window) displays the status of the current window stack. At the top of the window there is a menubar from which the user can select operations that transform the appearance of the focus. As can be seen in Figure 1 the window is divided into parts. Below the menubar there is a field that presents the relation that is preserved, which in Figure 1 is equality. There is one subwindow (the top one) that presents a pretty printed version of the current focus. The pretty printer associated with the environment has some specific features, which are explained below. The term in the window consists of subterms which can be accessed through their path. Normally manipulating a subterm would mean typing the operation to be performed and supplying the path to the subterm in question, which can be a tedious task. In TkWinHOL transformations are done differently. Double-clicking the right mouse button at some position in the term will highlight the smallest enclosing subterm that contains that position. After this, the user can select the operation wanted from the menus (or a button in some cases) and no path information is requested. A typical operation is opening a window. The user selects, by clicking, the subterm to open a window on and then press the button labeled OpenWindow (which can be seen at the bottom of the focus window in Figure 1).

To illustrate the way this works we can again refer to Figure 1. If the user clicks the `x` in the focus then only the `x` is selected (and highlighted). This is because `x` is a subterm on which it would make sense to open a subwindow. If the user clicks the leftmost `+` in the focus, then the selected subterm is `(x + y)`. Similarily, if the user clicks the middle `+` the whole focus is selected.

So in a sense TkWinHOL is, as for example Thery's chol [19], built around the structure of terms. However, this is not the only way to operate on the focus. Often there are situations where the structure of the focus is such that one cannot open a window on the part of the focus that one wants to, without modifying the focus first. Typically, such situations occur with associative operators where the standard association may not be what one wants. To simplify the users work in such situations TkWinHOL provides a way to select (by dragging, not clicking) a segment of the string representing the focus, and then opening a window on that selection although it may not be accessible using a path in the ordinary sense.

This is made possible by a routine that automatically reassociates the focus according to what the user wants and then opens a window on the subterm the user has selected. There are restrictions on the use of this feature; the text segment selected by dragging has to be of the correct form in order for reassociation to work.

We again refer to Figure 1 to illustrate how this works. As seen from the figure, the segment that has been selected (by dragging) is `y) + (z`. This is not a subterm in itself and cannot be selected by double-clicking as described above, but it is still possible to press the OpenWindow button which will result in a new window with the focus `y + z` (see Figure 2).

The subwindow at the bottom of the focus window contains the assumptions (hypotheses, conjectures and lemmas in window inference jargon) of the current stack. Operations that work on these are supported by TkWinHOL; users can click on the term that they want to operate on after which the term is highlighted. This term can then be used (as a theorem) as an argument to any command chosen from a menu. This saves the user from the trouble of typing the term (and additional type information).

There are also window commands in the menu bar that do not expect parameters of the kind described above. Parameters to such commands are passed via dialogue boxes or by selecting a string for example in the text editor window. Strings can also be selected in an emacs window if the user prefers to work in such a way. It is also always possible to modify the current window stack by issuing commands to the HOL window as described above. In this case the result of the issued command appears in the HOL window. The state of the focus window can be updated by pressing the ShowStack button.

3.2 Implementing TkWinHOL

As mentioned in Section 1 the main motivation for implementing TkWinHOL is to use it as a building block for something larger; an environment supporting formal program derivation in the Refinement Calculus framework. There were initial attempts to use both the Centaur system (which is used also in the chol system [19]) and the Cornell Synthesizer Generator [16]. Both these systems make it possible to build syntax directed editors for constructing programs in some user defined language. Both systems also allow defining transformation rules that operate on the programs (or more generally some syntactically correct term in a given language). However, the binaries that these systems produce are very big. Keeping in mind that HOL90 itself is a big system this is not a nice feature. Furthermore, both systems (being specialised) have some built in assumptions of what can and what cannot be done.

We chose to use Tcl/Tk instead. Tcl/Tk is a general purpose programming system for building graphical user interfaces. A typical Tcl/Tk application is small. Tcl/Tk applications are interpreted but that does not generally affect their efficiency substantially. In fact, the overhead produced by adding TkWinHOL on top of HOL90 is negligible. Tcl/Tk applications are also easy to extend since the language is easy to learn and additions are simply loaded into the application, no recompilation or linking etc. is needed.

To connect the interface to HOL we use an extension to the Tcl/Tk system called expect. This extension provides a set of functions lacking from Tcl/Tk, like job control over several processes running in the background simultaneously, and communicating with standard command line based application like HOL. Using expect makes it easy to run, for example, a computer algebra system simultaneously with HOL under the same interface. There are situations where this possibility could prove useful [11].

TkWinHOL relies on a special purpose pretty printer implemented in SML. Besides pretty printing terms this pretty printer also supplies information about the subterms of the pretty printed term. This information is needed to implement the selection of subterms by clicking the mouse.

3.3 Customising TkWinHOL

In this section we briefly describe how one can customise TkWinHOL for some specific HOL theory. A more detailed description can be found in [13]. There are basically three things one might want to do: modify the basic pretty printer supplied with the environment, produce a parser suited for the theory used and add new choices to the menus or even entire new menus. A parser is useful in the situations where one has to enter terms as arguments to menu choices.

As mentioned above the format the the pretty printer must have is quite simple. First of all it has to supply the TkWinHOL interface with information

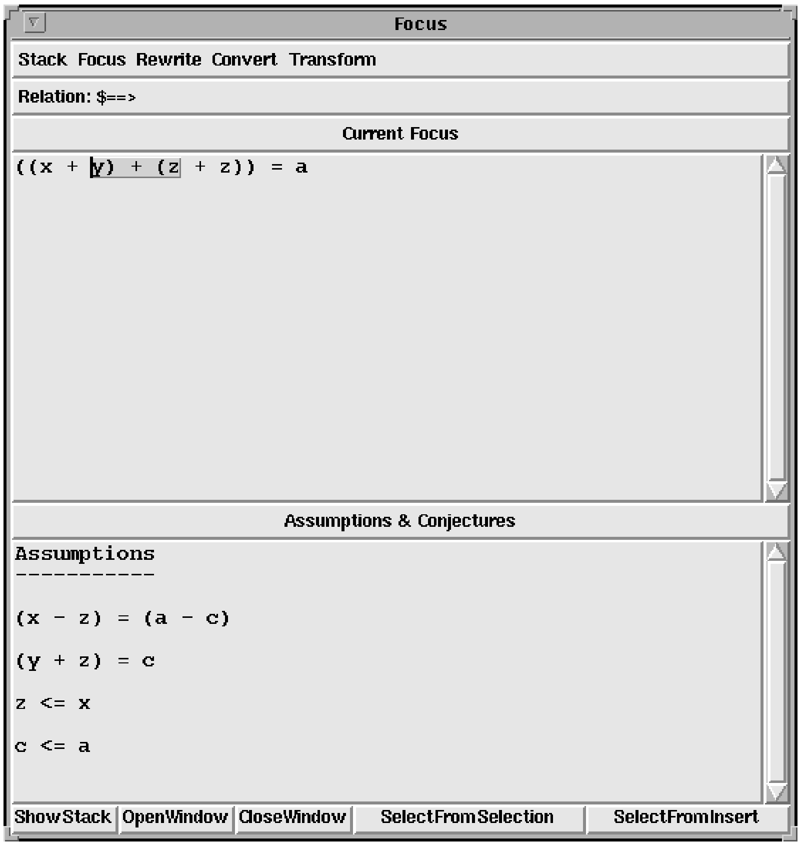

Figure 1: A sample view of the focus window of TkWinHOL

about the start and end point within the term (using a specific row + column format and path information) of each accessible subterm (besides the actual pretty printing). It is, however, not necessary to replace the basic pretty printer. It is only in the case that one wants the output to look 'nice' that a pretty printer is needed. A parser is also only needed in case one is not satisfied with the 'HOL-syntax' of the theory one uses.

Adding new choices in the menus or altogether new menus to the menubar is done using Tcl/Tk programming. In [13] examples are given how this can be done.

4 Using TkWinHOL – an example

We shall now illustrate how TkWinHOL supports transformations in a simple and intuitively appealing way. In particular we want to show features dealing with selection and regrouping of expressions. Selection is always essential and automatic regrouping substantially simplifes derivations that handle associative operators.

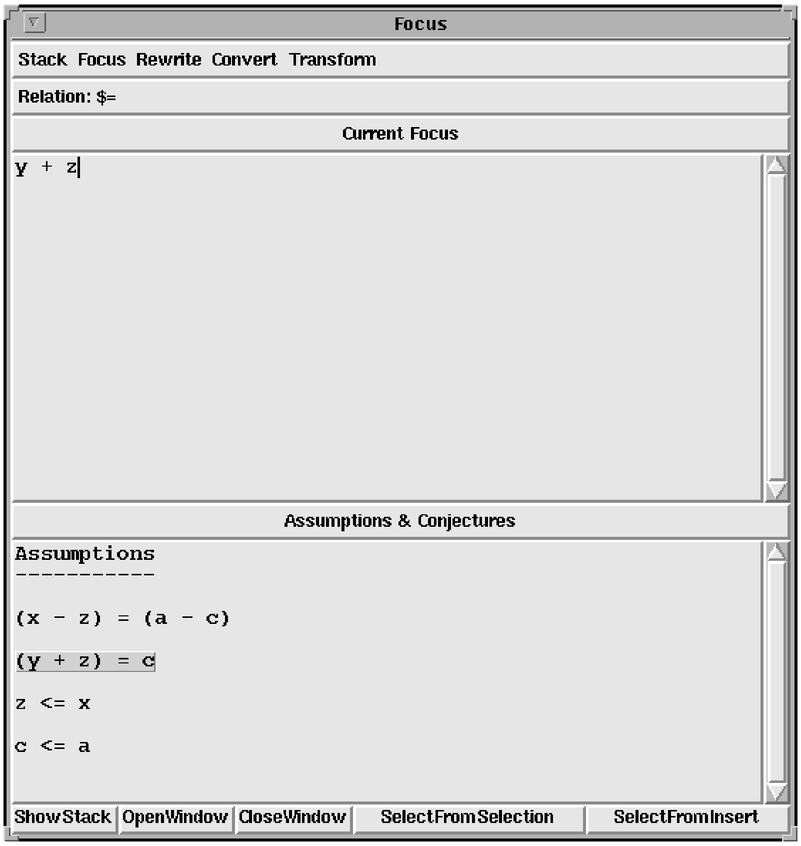

Figure 2: A sample view of the focus window of TkWinHOL

Suppose that we are to find a solution for a simple system of linear equations

$$\begin{array}{rcl} (x+y)+2*z & = & a \\ x-z & = & a-c \\ y+z & = & c \end{array}$$

under the assumptions $z \leq x$ and $c \leq a$, where x, y, z are variables and a, c stand for constants. We start by creating a stack containing the two assumptions and a conjunction of the above three equations as the initial focus. This is achieved by choosing the BEGIN_STACK menu entry with the arguments:

```
"equations"
(--`($==>) (((x+y)+2*z=a) /\ (x-z=a-c) /\ (y+z=c))`--)
[--`c<=a`--,--`z<=x`--] []
```

Our goal is to transform (preserving implication) it into a conjunction of three equalities with the right-hand side defining values of the variables on the left-hand side.

We begin by focusing on the first equation (conjunct). This can be done by double-clicking it (on equality symbol) and then pushing OpenWindow button.

Two other equations now become the assumptions of the current focus and can be used for its transformation. In particular the third equation `(y + z) = c` allows simplification of the focus. We only need to rewrite `2 * z` (using the REWRITE_WIN menu entry with the appropriate theorem) as a sum `z + z`. The result of all these transformations can be seen in the Figure 1. However we still have a little trouble since there is no (due to the different association) subterm `y + z` in the focus. In the plain window inference system one would need to rewrite the focus twice using the associativity theorem in order to get it. Here, however, we can proceed just by dragging the pointer and highlighting the desired subexpression (see Figure 1). Pushing OpenWindow button then automatically rearranges the current focus and as a result returns a new focus `y + z`.

Now we select by double-clicking the second assumption `(y + z) = c` in the assumption list (see Figure 2) and supply it as an argument for the menu entry REWRITE_WIN. TkHolWin then rewrites the focus with this assumption. Closing the window with the button CloseWindow gives us the new focus:

```
(x + (c + z)) = a
```

Note that in this way we can easily choose the assumptions to be used in a transformation and avoid unnecessary (and possibly looping) rewriting which occurs when directly using functions from the window inference library such as ASM_REWRITE_WIN.

At this stage we can notice close relation between the focus (first equation) and the first assumption (second equation). To exploit it we have to move variable `c` from the left to right-hand side of the equation in the focus. This can done using the theorem ADD_EQ_SUB:

```
|- !m n p. n <= p ==> ((m + n = p) = m = p - n)
```

However first the focus must be regrouped to contain subexpression `x + z`. We once rewrite the expression `c + z` with the commutativity theorem for addition then the focus is regrouped into new one:

```
((x + z) + c) = a
```

in the same way as described above (first dragging the pointer over `x + (z` then opening and closing window on it). Note that the theorem ADD_EQ_SUB is conditional, i. e. it can be used for rewriting only in the appropriate context. In our example the context would be `c <= a`. As we can see there is such assumption in the assumption list so we can choose the menu entry CONDITIONAL REWRITE and supply the theorem as an argument. TkWinHOL automatically checks whether rewrite condition is satisfied and then rewrites the focus.

Untill now we have transformed the focus into a form which allows to apply the lemma (it could be derived in a similar way):

```
|- !p m n. n <= m ==>
           (m + n = p) /\ (m - n = p) ==> (m = p) /\ (n = 0)
```

to the first two equations. We proceed by closing the window and regrouping as above the (conjunction of) equations to focus on the first two:

```
((x + z) = (a - c)) /\ ((x - z) = (a - c))
```

It is easily seen that the focus matches the left-hand side of the right implication in the lemma. So we instantiate the lemma and use it as an argument for the transformation choosen with the TRANSFORM FOCUS menu entry. TkWinHOL checks first whether (i) the context condition (in our case `z <= x`) is satisfied, (ii) the relation to be preserved in the derivation (implication) is the same as in the transformational theorem, and then transforms the focus into

```
(x = (a - c)) /\ (z = 0)
```

which gives the values of the variables x and z.

Now we close the old window and open the focus on the third equation `(y + z) = c`. The new assumption `z = 0` becomes applicable in the rewriting and can be selected for it by double-clicking. We finish the derivation by rewriting the focus with the theorem ADD_0 which says `!m.  m + 0 = m` and closing the window. The current focus

```
((x = (a - c)) /\ (z = 0)) /\ (y = c)
```

is the solution for the initial equation system (under the given assumptions). This fact is expressed by the window theorem:

$$
\begin{array}{ll}
 & c \leq a, z \leq x \\
\vdash & \\
 & (x + y) + 2 * z = a \wedge x - z = a - c \wedge y + z = c \\
\Rightarrow & \\
 & x = a - c \wedge z = 0 \wedge y = c
\end{array}
$$

which shows the result of the derivation described here.

5 The Refinement Calculator

This section briefly describes, through an example, how TkWinHOL has been customized into a Refinement Calculator – a tool for program refinement.

Program refinement The Refinement Calculator is based on a mechanisation of the Refinement Calculus in HOL [12]. In the Refinement Calculus, specifications are refined into programs through a sequence of transformation which preserve a *refinement* relation (preserving total correctness, see [3, 4, 6] for details). In the example below, we use a subset of the Refinement Calculus laguage with an intuitively simple semantics. Essentially, this is Dijkstra's guarded commands extended with assertions and with a nondeterministic assignment used for specification purposes.

The Refinement Calculator extends TkWinHOL with a number of menu alternatives for refinement transformations. For each transformation, HOL proves the correctness using the underlying theory of refinement. The example below illustrates a few basic transformations. More advanced transformations are also available, such as introduction of blocks with local variables, recursion introduction and data refinement.

The Refinement Calculator contains a parser (written by Michael Butler) which parses program statements entered by the user into the abstract syntax of the HOL system. It also contains an extension of the TkWinHOL pretty-printer to handle program statements.

A small refinement example We illustrate the Refinement Calculator by showing a derivation of a program that computes a factorial. We begin a program derivation by entering the initial specification

```
"program ex var x,y:num . {y=N}; x,y:=x',y'. x' = FACT N"
```

A special menu alternative for beginning derivations sends the string to a parser for the Refinement Calculus notation. The Window Inference system then begins a new stack and returns the program term which is pretty-printed in the focus window. The specification has name `ex` and it contains an assertion and a nondeterministic assignment. The assertion gives the precondition `y=N` and the nondeterministic assignment says the final value of `x` is `N!` while the final value of `y` is not constrained. Here `N` is a specification constant, not a program variable.

Our first transformation step is to add an initialisation to `x`. We open a subwindow (by pointing and clicking) on the nondeterministic assignment and select the refinement-specific menu alternative ADD ASSIGNMENT with argument `"x:=1"`. After this, we close and then use a special menu alternative PUSH ASSERT to propagate the assertion through the assignment. Closing after this, the focus is

```
{ y = N };
x := 1;
{ (x = 1) /\ (y = N) };
x,y := x',y' . x' = (FACT N)
```

So far, the correctness of each refinement step has been proved automatically by the Refinement Calculator (i.e., by HOL in the background).

Now we want to replace the combination of an assertion and a nondeterministic assignment with a do-loop. We open a subwindow on this statement pair and select a LOOP INTRODUCTION menu alternative from the Transform menu and give the arguments (guard, body, invariant and termination function)

```
"0<y"
"x,y:=x*y,y-1"
"FACT N = x*(FACT y)"
"y"
```

in the dialogue box which appears.

The Refinement Calculator replaces the focus with the statement

```
do 0<y -> x,y := x*y,y-1 od
```

and adds three proof obligations which are shown in the assumption subwindow. These proof obligations can be proved immediately or later, also using the Refinement Calculator. In this simple example, one is proved automatically, one requires only trivial arithmetic and one is a simple total correctness assertion which is proved using basic properties of the factorial function.

Finally, we can close and use a special menu alternative to drop assertions from the program text (again, correctness proofs of the transformations are automatic). This leaves us with the final program for computing a factorial

```
x := 1;
do 0<y -> x,y := x*y, y-1 od
```

which is thus a guaranteed correct refinement of the original specification.

6 Conclusions

We have described a graphical interface to the HOL Window Inference library. The main inspiration for developing it was to provide a user friendly tool for transformational reasoning. The main aim was to automate very simple but nevertheless tedious steps used in window inference such as selecting a subexpression or regrouping an expression into appropriate form. We have provided facilities which allow these steps to be carried out in a simple way. An expression can be selected by pointing on appropriate part of it and clicking a mouse button. Some (part of) expression can be regrouped using associativity properties of the operators just by selecting segment of the text which we are interested in.

This simple way to select 'interesting' (sub)expressions in the focus window allows easy selection of the objects to be manipulated in some transformation (for instance, selecting an assumption to be used in rewriting). In the future, a multiply selection facility will make this feature even more powerful. Note that the user neither needs to enter HOL terms nor supply any extra information (types, numbers of assumptions, etc.) concerning the objects of interest.

We have also provided an example of how TkWinHOL can be used for more specific purposes: it served as a basis for an environment supporting formal program derivation within the Refinement Calculus framework.

The choice of the Tcl/Tk programming system (with the expect extension) for implementing the tool turned out to be a good one. It made our interface customisible and easily extensible with new functionality and new menu choices. The Refinement Calculator is a good illustration of this.

Acknowledgements

We wish to thank Michael Butler who implemented the parser for the Refinement Calculator Jim Grundy who implemented the Window Inference Library and also commented on a draft version of this paper.

References

[1] F. Andersen. *A Theorem Prover for UNITY in Higher Order Logic*. PhD thesis, Technical University of Denmark, Lyngby, 1992.

[2] Flemming Andersen, Kim Dam Petersen, and Jimmi S. Petterson. A Graphical Tool for Proving UNITY Progress. In T. F. Melham and J. Camilleri, editors, *Higher Order Logic Theorem Proving and Its Applications – 7th International Workshop. Valletta, Malta, September 1994*, volume 859 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[3] R.J.R. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications*, volume 131 of *Mathematical Center Tracts*. Mathematical Centre, Amsterdam, 1980.

[4] R.J.R. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988.

[5] R. J. R. Back and J. von Wright. Refinement concepts formalized in higher order logic. *Formal Aspects of Computing*, 2:247–272, 1990.

[6] R.J.R. Back and J. von Wright. Refinement calculus, part I: Sequential programs. In *REX Workshop for Refinement of Distributed Systems,* volume 430 of *Lecture Notes in Computer Science*, Nijmegen, The Netherlands, 1989. Springer–Verlag.

[7] A. Camillieri. Mechanizing CSP trace theory in Higher Order Logic. *IEEE Transactions on Software Engineering*, 16(9):993–1004, 1990.

[8] M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL*. Cambridge University Press, 1993.

[9] J. Grundy. A window inference tool for refinement. In Jones et al, editor, *Proc. 5th Refinement Workshop*, London, Jan. 1992. Springer–Verlag.

[10] J. Grundy. HOL90 window library manual. 1994.

[11] John Harrison and Laurent Théry. Extending the HOL Theorem Prover with a Computer Algebra System to Reason about the Reals. In Jeffrey J. Joyce and Carl-Johan H. Seger, editors, *Higher Order Logic Theorem Proving and Its Applications – 6th International Workshop, HUG '93 Vancouver, B. C., Canada, August 1993*, volume 780 of *Lecture Notes in Computer Science*, pages 174–184. Springer Verlag, 1993.

[12] P. Luostarinen J. von Wright, J. Hekanaho T. Långbacka. Mechanising some advanced refinement concepts. *Formal Methods in Systems Design*, 3:49–81, 1993.

[13] Thomas Långbacka. *TkWinHOL Users Guide*. In preparation.

[14] Don Libes. *Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs*. O'Reilly & Associates, 1994.

[15] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison–Wesley, 1994.

[16] Thomas W. Reps and Tim Teitelbaum, editors. *The Synthesizer Generator. A System for Constructing Language-Based Editors.* Springer-Verlag, 1988.

[17] P.J. Robinson and J. Staples. Formalising the hierarchical structure of practical mathematical reasoning. Techn. Rep. 138, Key Centre for Software Technology, University of Queensland, Australia, 1990.

[18] Donald Syme. A New Interface for HOL - Ideas, Issues and Implementation. submitted to the HUG95.

[19] Laurent Théry. A Proof Development System for the HOL Theorem Prover. In Jeffrey J. Joyce and Carl-Johan H. Seger, editors, *Higher Order Logic Theorem Proving and Its Applications – 6th International Workshop, HUG '93 Vancouver, B. C., Canada, August 1993*, volume 780 of *Lecture Notes in Computer Science*, pages 115–128. Springer Verlag, 1993.

[20] J. von Wright. Program refinement by theorem prover. In *BCS FACS Sixth Refinement Workshop – Theory and Practise of Formal Software Development. 5th – 7th January, City University, London, UK.*, 1994.