Adding External Decision Procedures to HOL90 Securely

Elsa L. Gunter

Bell Laboratories,
600 Mountain Ave.,
Murray Hill, NJ, 07974, USA
email: elsa@research.bell-labs.com

Abstract. This paper describes a modest conservative extension of HOL90 that allows the results from external decision procedures to be used within HOL90 without compromising its logical consistency.

1 Introduction

Theorem provers such as HOL90 place a great deal of emphasis on being expressive and on being secure. As a result, they are inherently interactive, sometimes to the annoyance of the user. On the other hand, decision procedures, such as BBD's and model checkers [5, 6], place a great deal of emphasis on being totally automatic and fast. However, they work on restricted languages and their security is usually checked by hand proof at most. Therefore, the level of trust that is reasonably put in their results may be somewhat less than the level of trust reasonably put in the results of theorem provers. To improve the level of automation of a theorem prover, it is sometimes desirable to call an appropriate decision procedure from within the theorem prover when the problem being worked on has been reduced to the appropriate subset. PVS [7] is an example of a theorem prover that makes use of such a link to model-checkers.

The problem arises, however, of how to incorporate the results of an arbitrary decision procedure within a theorem prover without compromising the security of the theorem prover. One solution to incorporating decision procedures in a fully expansive theorem prover, such as HOL90, is to write them as tactics, conversions, etc. This solution provides the highest security, but often leads to much less efficient procedures, since they must actually build a proof, not just decide whether one exists. Also, it doesn't allow us to directly take advantage of existing external decision procedures.

In this paper we will describe a mechanism for using external decision procedures from within HOL90 while maintaining HOL90's high standard of security. This method should be applicable to any of the family of HOL theorem provers, and most likely to a much braoder class than that. The basic idea is that each theorem will carry with it a tag indicating that an external result was used. This tag is internal to the logic; it is not an external annotation on theorems. There are two variants of tagging possible; one just documents the fact that an external procedure was called, and the other records in addition which results were accepted and used.

The version that records the results that were accepted provides the greatest documentation, and allows for the subsequent elimination of dependency on external results should they be proved within the theorem prover. The usefulness of this method will depend on the number of results from external decision procedures being relatively small. For those instances when the number of results from external decision procedures is quite large, we provided a more limited, but equally secure method for incorporation. Again the theorems using external results are in essence tagged, but only with that fact that an external procedure was used, and not with the result itself. In this second method, it will not be possible to eliminate the tag because the precise result assumed is not recorded.

2 The Library add_dec_proc

We have added to HOL90 a contributed library named add_dec_proc which supports two methods for allowing HOL90 to accept as theorems results from external decision procedures while maintaining the logical consistency of HOL90. This library consists of a theory dec_proc_tokens introducing a type and some constants used for tagging, and functions implementing two new primitive inference rules, corresponding tactics, and a modified version of the goalstack manipulation functions expand and e. We also include a trivial example defining the "decision procedure" clearly, used by humans when they don't wish to give the details of a proof, and a more substantive example using the method in conjunction with Peter Homeier's Verification Condition Generator [4].

To support the tagging in both methods, we introduce a type : dec_proc_token. The type : dec_proc_token is defined to be isomorphic to an arbitrary non-empty subset of the infinite type : ind. (We use the Hilbert choice operator here to select such a subset.) Elements of this type are treated as the names of the external decision procedures. They will be used as arguments to constants that yield boolean results that will become hypotheses of theorems using the external procedures in their proofs. Both of the methods we describe work basically the same way, and we shall describe them in sequence here.

The first of the two methods for accepting external results is somewhat simpler to understand, and more concise, but it is also of less utility. Let proc be the name of a decision procedure external to the HOL90. To be able to use the library with either of the two methods, the user must supply a procedure for calling proc and for notifying the HOL90 of the result. For the first method, we shall assume that we have an SML procedure `call_proc : term` $\rightarrow$ `unit` which returns `() : unit` if proc was able to verify the goal, and raises an `HOL_ERR` exception otherwise. By raising an exception, it can be used as a component of tactics that try various options depending upon which options succeed or raise an exception. Using new_constant, we may introduce a constant proc : dec_proc_token to represent the procedure within HOL90. (If the user does not wish to introduce their own constant, the library supplies the constant default_token : dec_proc_token which may be used instead.) With these two pieces, we may make use of the supplied SML function

```
trust_proc_result {token = (--`proc`--),
                   dec_proc = call_proc} : term -> thm
```

which implements the new primitive inference rule

$$\frac{}{[\mathsf{trusted\_token\ proc}] \vdash tm}$$

assuming `call_proc` tm completes without raising an exception. The hypothesis trusted_token proc intentionally may be thought of as an oracle saying whether proc always proves correct results. If it does, then the hypothesis is true and we have the desired theorem; if it does not always give correct results, then the theorem is still valid since the hypothesis is false — it is just meaningless. Extensionally, trusted_token is a constant introduced via constant specification to be of HOL90 type : dec_proc_token $\rightarrow$ bool, and the result of an application is either T or F, but we cannot in general know which. On the other hand, the primitive inference rule `trust_proc_result` is a proper logical extension to HOL90 and will require proof that it is a conservative one.

We now have the basic machinery for incorporating external results, at least for the simple method. However, typically we do not want to use `trust_proc_result` directly, but rather we would prefer to use a version that works as a tactic on goals on the goalstack. The corresponding tactic

```
trust_proc_result_TAC : {token = (--`proc`--),
                         dec_proc = call_proc} : tactic
```

calls `trust_proc_result` on the current goal rendered as a term. If the goal is $[a_1, \ldots, a_n] \vdash tm$ with free variables $x_1, \ldots, x_m$, then the term to which `trust_proc_result` will be applied is $\forall x_1 \ldots x_m.\ a_1 \wedge \cdots \wedge a_n \Rightarrow tm$.

The tactic `trust_proc_result_TAC` resolves the theorem resulting from the call to `trust_proc_result` (if there is one) with the assumptions of the goal, and attempts to derive the goal as a theorem. However, the theorem returned has a hypothesis not occurring (at least not usually) among the assumptions of the goal, namely trusted_token proc. The standard functions expand and e for applying tactics to the goalstack perform a validity check which assures that all the hypotheses of the theorem returned by the validation computed by the tactic are among the assumptions of the current goal. Therefore, the tactic `trust_proc_result_TAC` will be rejected by these standard goalstack manipulation functions. To address this, we have included a modified version of these functions. The modified version of these two functions performs almost the same validity checks as before; the one difference is that now they ignore tagged hypotheses (from either method) among the hypotheses of the validation theorem. With this modification, `trust_proc_result_TAC` can be used the same way as other tactics. This pretty much completes the story (accept for proof of consistency) for the simple method.

The second method has the same components as the simple method, but in a more complicated, and informative manner. There are also some additional functions to make use of some of the added flexibility of this second method. The tokens generated to name external procedures for the simple method may be reused for the second method. However, this time the user is expected to supply an SML function for calling the procedure that returns a term stating the theorem to be accepted, if there is one. Thus, in our case, we will assume we have `consult_proc : term` $\rightarrow$ `term`. As before, we may now create

```
use_proc_result {token = (--`proc`--),
                 dec_proc = consult_proc} : term -> thm
```

which, when applied to the input term *in_term*, implements the primitive inference rule

$$\frac{}{[\mathsf{used\_token\ proc}\ tm] \vdash tm} \quad \text{where } \texttt{consult\_proc}\ \mathit{in\_term} = tm$$

provided the term *tm* returned by `consult_proc` is a closed term, *i.e.* contains no free variables. This inference rule is not only more informative (by putting the result accepted from and external procedure in the hypotheses), but it is also more broadly applicable than the one given by `trust_proc_result`. The rule can be used to couple HOL90 with calculators (such as computer algebra systems), as well as decision procedures. For example, it could be used to couple HOL90 with the Unix utility `dc`. In that case, the term to which it would be applied would be a complex numeric expression and the result would be a term stating that the expression was equal to its numeric value.

It will oftentimes be the case that we do not want to make use of this extra flexibility. Then we would like to have just one method for calling a given external procedure. To facilitate using one kind of calling function where the other is required, we supply two coercion functions: `trust_use` and `use_trust`. Given `call_proc` used with the simple method, we could define `consult_proc` by:

```
val consult_proc = trust_use call_proc
```

The term that will be returned is the universal closure of the term to which `consult_proc` is applied. Similarly, if we have `consult_proc` and we wish to define `call_proc`, we could use

```
val call_proc = use_trust consult_proc
```

This function will only succeed if the term returned by `consult_proc` is the universal closure of the term to which `call_proc`, and hence `consult_proc`, is applied.

As in the first method, the way we expect `trust_proc_result` to be most commonly used is not directly, but through a tactic. In this situation, however, it makes sense to apply the tactic only to calling functions that return the (universal closure of) the term to which they are applied. This is reflected by the type of the decision procedure calling function argument being the same as it is for the tactic `trust_proc_result_TAC`. The new tactic we get is:

```
use_proc_result_TAC {token = (--`proc`--),
                     dec_proc = use_trust consult_proc} : tactic
```

which calls `use_proc_result` on the current goal. It, too, resolves the resulting theorem (if there is one) with the assumptions of the goal and attempts to derive the goal as a theorem.

The inference rule given by `use_proc_result` introduces a hypothesis, much as `trust_proc_result` does. The nature of this hypothesis is different, however, in that it contains the theorem statement, tm, as a subterm. This causes some difficulty with the interaction with tactics. We restricted the decision procedure calling functions to ones that return closed terms because it makes the tactic `use_proc_result_TAC` operate more robustly in conjunction with other tactics. Without this restriction, proofs built with `use_proc_result_TAC` and tactics such as `GEN_TAC` which introduce scoped free variables would fail when the final theorem was being built because there would be free variables in the hypotheses introduced by `use_proc_result_TAC` that would need to escape their scope. There is a similar concern with type variables, but without the ability to quantify propositions by type variables, we are limited in our ability to protect against it. Whenever `use_proc_result_TAC` is used on a goal where the validating theorem generated by `use_proc_result` contains type variables, a warning message is printed.

As we have seen above, the fact that `use_proc_result` introduces a hypothesis with the theorem statement causes some difficulty with tactic style theorem proving. Therefore, the question arises of why we would want this version over what the simple method gives us. There are two answers to this question: documentation and future elimination. The hypotheses serve as documentation, recording for each theorem all the results that depended on theorems from `use_proc_result`. Moreover, since the result accepted from external procedures are carried along with the theorems they go into proving, if we were to prove those results in HOL90, then roughly by Modus Ponens we ought to be able to eliminate them from the hypotheses of any theorem. And in fact we can. This is given to us by the way used_token is introduced. The constant used_token is specified to satisfy

$$\vdash \forall tok\ p\ q.\ p \wedge ((\mathsf{used\_token}\ tok\ p) \Rightarrow q) \Rightarrow q.$$

Constant specification requires us to show that there exists a value that satisfies the given property. In this case, we can use the function $\lambda\, tok\, p\, .\, p$ as our witness. This specification gives us the desired elimination property.

We have the specification of used_token, but how are we to think of it? Intentionally, we would like to think of it as telling for each proposition to which it is applied whether it was correctly verified by the associated external procedure. Unfortunately, that doesn't really match with the extensional view. Extensionally, each proposition is either equal to T or F. Therefore, if used_token proc returns T for any true proposition, then it must return T for all true propositions. Thus, probably the best intentional understanding of used_token proc is that it is the identity function. There is one other extensional possibility for $\mathsf{used\_token}\ \mathsf{proc}$ which we shall discuss in Section 4.

In this second method we saw that we can use `use_proc_result` in conjunction with various calculators to generate theorems simplifying expressions or perhaps solving for unknowns. It seems unnecessarily confining to require all theorems that make use of such calculations to carry the results of the calculations with them as hypotheses, as opposed to carrying the simpler tag of the first method. Space considerations may render such record-keeping impractical. In which case, we would like to have the same functionality, but with the simpler tagging. This is given to us by the property used for the constant specification of $\mathsf{trusted\_token}$, which we failed to give earlier. That specification is

$$\vdash \forall tok\ p\ q.((\mathsf{used\_token}\ tok\ p) \Rightarrow q) \Rightarrow ((\mathsf{trusted\_token}\ tok) \Rightarrow q).$$

The witness that makes this a legitimate constant specification of $\mathsf{trusted\_token}$ is $\lambda\ tok\ .\ \mathsf{F}$. This specification allows us to replace any hypothesis of the form $\mathsf{used\_token}\ tok\ p$ by ones of the form $\mathsf{trusted\_token}\ tok$. To automate this we have the SML function `used_hyp_to_trusted_hyp : term -> thm -> thm` which implements the derived rule of inference:

$$\frac{[\mathsf{used\_token}\ tok\ p, a_2, \ldots, a_n] \vdash q}{[\mathsf{trusted\_token}\ tok, a_2, \ldots, a_n] \vdash q}$$

3 Examples

To illustrate the usefulness of the library $\mathsf{add\_dec\_proc}$, we have created two examples of its application. As a simple example of how these pieces fit together, we have written a trivial example "decision procedure" called `clearly`. Given a term tm, `clearly` will return the term $\forall x_1 \ldots x_m.tm$ where $x_1 \ldots x_m$ are all the free variables in the term. Naturally, this procedure decides nothing. It is used only when the person proving a theorem wants to quit at some level. This can be a useful thing to do in some cases. In addition, we have introduced a token $\mathsf{clearly}$ to represent this procedure in HOL90. Given `clearly` we can then define the rule

```
val clearly_RULE =
    use_proc_result {token = (--`clearly`--),
                     dec_proc = clearly} : term -> thm
```

which is in essence `mk_thm` but on closed terms instead of sequents, and the tactic

```
val clearly_TAC =
    use_proc_result_TAC {token = (--`clearly`--),
                         dec_proc = use_trust clearly} : tactic
```

which solves any goal. In this form, `clearly_TAC` can be a quite useful tactic, allowing one to postpone the completion of certain proof obligations indefinitely, while still retaining the ability to discharge them at any time should one happen to actually prove them. We feel that `clearly_RULE` and `clearly_TAC` are general enough and useful enough that they have been included as part of the library add_dec_proc.

The second example is a modification of Peter Homeier's Verification Condition Generator for the Sunrise system [4], as included in the `contrib` library `vcg` for HOL90. This example is included as a separate file, and depends upon both the `add_dec_proc` library and the `vcg` library. The library `vcg` gives a verification condition generator for a small imperative programming language, Sunrise, with mutually recursive procedures, proving total correctness. This is implemented two ways, one where the verification condition generator has been implemented as conversions and tactics within HOL90, and the other where the basic programs for checking well-formedness and for generating the verification conditions are written in SML, with their results accepted into HOL90 by the use of `mk_thm`. We shall refer to the second approach as the "fast" approach and the first as the "secure" approach. The second approach is considered unsound in principle, but the algorithms for these two functions were verified as part of the project, and thus in this instance it would be reasonable to assume that the second method is roughly as secure as the method that does all the proof in HOL90. We have rewritten the conversions `FAST_WFp_CONV` and `FAST_vcg_CONV` from the fast version so that instead of calling `mk_thm`, they use `use_proc_result`. We have then added a derived rule of inference `VCG_SIMPLIFY` that uses the conversions `WFp_CONV` and `vcg_CONV` from the secure version to eliminate the tagged hypotheses giving well-formedness and stating what the verification conditions are (or rather, that they are sufficient). By modifying this work in this manner we make it possible for proofs of program correctness to be done interactively using the fast version, when the human user doesn't want to wait, and then later to clean up to totally verified proofs by doing the well-formedness and vcg results totally automatically after the fact. While this division may not be especially useful for proofs in the Sunrise system, where the verification condition generator has actually been verified, it does illustrate a methodology that can be applied to similar projects where security is critical, but where there is not the time to carry out such a system verification. While in this method the user must still incur the cost of actually proving the well-formedness and vcg results generated by the fast version, it is done so at minimum cost with no reproving required, and it can be done after all interactive parts have been completed, off-line as it were.

It is worth noting that this example also takes full advantage of the additional flexibility of `use_proc_result` over `trust_proc_result`. The conversion `FAST_WFp_CONV` is in essence a decision procedure, and as such could have been implemented using `trust_proc_result` had we not been interested in eliminating our dependence on its results. However, `FAST_vcg_CONV` is really a calculation of the verification conditions, which are not known in advance. Therefore, we need `use_proc_result` to hand back a term telling us what those conditions are.

4 Consistency with the existing system

The library $\mathsf{add\_dec\_proc}$ is a proper extension of HOL90; it adds two new primitive inference rules `trust_proc_result` and `use_proc_result`. The question arises: Why is this a logically sound thing to do? The rules `trust_proc_result` and `use_proc_result` may be seen as introducing a family of axioms, all of the form

$$\frac{}{[\mathsf{trusted\_token\ tok}] \vdash P} \quad \text{or} \quad \frac{}{[\mathsf{used\_token\ proc}\ tok] \vdash P}$$

for some constant tok and some proposition P. These may all be seen as specific instances of the propositions

$$\forall tok\ P. \mathsf{trusted\_token}\ tok \Rightarrow P \tag{1}$$

$$\forall tok\ P. \mathsf{used\_token}\ tok\ P \Rightarrow P. \tag{2}$$

The only other axioms we have about $\mathsf{trusted\_token}$ and $\mathsf{used\_token}$ are their specifications:

$$\vdash \forall tok\ p\ q.\ p \wedge ((\mathsf{trusted\_token}\ tok\ p) \Rightarrow q) \Rightarrow q \tag{3}$$

$$\vdash \forall tok\ p\ q.((\mathsf{used\_token}\ tok\ p) \Rightarrow q) \Rightarrow ((\mathsf{trusted\_token}\ tok) \Rightarrow q). \tag{4}$$

The propositions (1), (2), (3) and (4) are all satisfied if $\mathsf{trusted\_token}$ is defined to be $\lambda\ tok\ .\ \mathsf{F}$ and $\mathsf{used\_token}$ is defined to be $\lambda tok\ prop.\ prop$, i.e., essentially the identity function. We can make these as definitions in HOL90 without any new extensions, and derive the propositions (1), (2), (3) and (4) as theorems. Since any definitional extension to HOL90 is known to be conservative, to see that the extension given in this paper is a conservative extension, it suffices to show that any theorem of the new system is also a theorem in the system given by the definitional extension. A rigorous proof of this is done by induction on the height of the proof tree (given as a sequent style encoding of natural deduction proofs) of a theorem in the new extension. The only cases of interest are the base cases. If we make use of one of the new primitive rules of inference or one of the axioms of constant specification in the new extension, they must be replaced by the derived results in the definitional extension. From here all applications of inference rules translate directly without modification.

Let us consider the possible semantics in HOL90 of $\mathsf{trusted\_token}$ and $\mathsf{used\_token}$. Because of their types, for each token tok, there are two possible values for $\mathsf{trusted\_token}\ tok$ and four possible functions for $\mathsf{used\_token}\ tok$. As indicated above, it is consistent with the extended system to interpret $\mathsf{trusted\_token}$ as $\lambda\ tok\ .\ \mathsf{F}$ and $\mathsf{used\_token}$ as $\lambda tok\ prop.\ prop$. For each token tok for which only true results have been returned, it is also possible to interpret $\mathsf{trusted\_token}\ tok$ as the value T. As soon as a given procedure proc accepts a false result (for either `trust_proc_result` or `use_proc_result`), from that point on the only valid interpretation of $\mathsf{trusted\_token\ proc}$ is F. Also as indicated above, it is consistent to interpret $\mathsf{trusted\_token}$ as $\lambda\ tok\ .\ \mathsf{T}$. The other three possibilities are that it maps everything to T, that it maps everything to F, or that it is negation. However, because we have the axiom

$$\vdash \forall tok\ p\ q.\ p \wedge ((\mathsf{used\_token}\ tok\ p) \Rightarrow q) \Rightarrow q$$

we must have that $\mathsf{used\_token}\ tok\ \mathsf{T}\ =\ \mathsf{T}$. Thus $\mathsf{trusted\_token}\ tok$ could be the identity function, or it could map both T and F to T. As long as `use_proc_result` only returns theorems with conclusions which are true, it is also consistent to interpret it as either of these functions. However, for each procedure proc for which $\mathsf{mk\_trusted\_thm}$ has returned a theorem

$$[\mathsf{used\_tokenproc}\ P] \vdash P$$

where P is provably equal to F, we must have that $\mathsf{trusted\_tokenproc}$ is the identity function. As long as our decision procedures never return false results, there will remain multiple interpretations (two for each token) of both $\mathsf{trusted\_token}$ and $\mathsf{used\_token}$.

5 Future Work

The library described in the paper is largely untested. The next step is to build a class of decision procedures in SML and hooks through SML to other independent procedures to be used with this library and to carry out realistic examples using them. One way in which we will create the decision procedures in SML will be to choose an existing implementation of some decision procedure, such as a model checker, translate HOL90 terms into the syntax accepted by that implementation, pipe the appropriate string into it, and collect the response. After creating a few such procedures, it may become clear whether there is additional common infrastructure that is desired.

6 Related Work

The methods described in the paper allow results from external sources to be incorporated in HOL90 as theorems, but with tagged hypotheses. In his 1992 HOL conference paper [1], Richard Boulton presented a method of achieving much the same effect by creating and additional datatype of `lazy_thm`. This was a method external to the logic and required a fair amount of duplication of functions for theorems as functions for lazy theorems. John Harrison went on to use this method in his work coupling HOL with computer algebra systems [3]. The advantage of Richard Boulton's work is that it requires no change to the logic. Our work does require a change to the logic, but is it provably consistent and we feel is actually much more light-weight. It also provides many of the advantages of his system; implementing John Harrison's work should be entirely straightforward in this new method, for example.

In a recent release of Konrad Slind's system HOL98, the core data structure for theorems has been changed to carry tags to support the inclusion of results from external procedures. Once a result is obtained from an external source it will be tagged and the tag will appear in all theorems subsequently derived from the result. This method provides essentially the same functionality as the first method described in this paper. And it suffers the same limitation in its inability to eliminate tags once they have been introduced. Moreover, it carries a greater overhead with it than our first method does, since every theorem must have a tag field, and every step of inference must merge the tag fields of the input theorems, even when those fields are empty. In both versions of our method, no additional overhead is incurred for those theorems whose proofs are done entirely within the system.

In the most recent release of Isabelle, oracles have been added. It appears to use a mechanism quite similar to that of HOL98. We believe that the PVS system has some mechanism, possibly similar to lazy theorems, but we have not seen it formally described in the literature.

Acknowledgements

I would like to thank Amy Felty, Carl Gunter and Davor Obradovic for their helpful discussions on this topic.

References

1. R. J. Boulton. A Lazy Approach to Fully-Expansive Theorem Proving. In *Higher Order Logic Theorem Proving and Its Applications*. North-Holland, 1992.
2. Gordon, M.J.C. and Melham, T. (1993) *Introduction to HOL*. Cambridge University Press.
3. J. Harrison and L. Théry. Extending the HOL theorem prover with a Computer Algebra System to Reason about the Reals. In *Higher Order Logic Theorem Proving and Its Applications*. Springer-Verlag, 1993.
4. P. V. Homeier and D. F. Martin. A Mechanically Verified Verification Condition Generator. In *The Computer Journal*, Vol. 38, No. 2, July 1995, pages 131-141.
5. G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall Software Series, 1991.
6. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
7. S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In *Proceedings of CAV'96*, Lecture Notes in Computer Science. Springer Verlag, 1996.
8. L. C. Paulson. The Isabelle Reference Manual. `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle98/doc/ref.dvi`
9. K. Schneider, R. Kuma, and T. Kropf. Integrating a first-order automatic prover in the HOL environment. In *Proceedings of the 1991 International Tutorial and Workshop on the HOL Theorem Proving System and Its Applications*. IEEE Computer Society Press, 1992.