H

By: Roberto Ghislanzoni (Roberto Ghislanzoni:MKT:RXI)

INTRODUCTION

The package H allows the user to augment the Interlisp–D environment with an Horn Clauses Theorem prover: in it it's possible to call semantic attachments (SAs) to Lisp (i.e. lisp functions) as FOL does.

A BRIEF HISTORY

The original idea comes from Chester (Chester,1980), (Chester, 1980): it was revised by Simmons (Simmons, 1984). A prototype of this program was developed at the University of Milan by Vieri Samek Ludovici, Giorgio Tornielli and Roberto Ghislanzoni using VLisp. Currently it is offered on Interlisp–D environment.

USE OF PACKAGE

Load the file H-LOAD.DCOM. This loads all necessary files. On your machine, you have now two environments: the H developer and the H deliver. In H developer you can plan, construct and edit your HKBs (H Knowledge Bases), and use it from Lisp executive window, simply by H.loading the HKBs created. H is a very good logic paradigm for LOOPS: from executive window, it is possible to have as many calls to the prover as you need.

THE H DEVELOPER

From the background menu chose H: this offer you a window in which is active a *read-prove-print* loop. Open as many windows as you want: all HKBs are local to the windows. The H Control Window has the following entries:

— **Show Profile** : This shows in the Prompt Window the current settings for environment bound to the window; the MODE of demonstration (FIRST: stop to the first goal proved; ALL: reach all goals: T: interactive mode); the LIMIT of the search tree; the TRACING of prover: the PMTRACING, that shows the pattern matching at work.

— **Show(Axiom)** : shows the clauses that define a predicate; the submenu allows to see the lambda-definition of a semantic attachment. The choice is made from a pop-up menu.

— **Delete(Axiom)** : this erases from database the clauses choosen by the user. Erases also the SA from the submenu.

— **Edit(Axiom)** : it allows to create and edit both predicates and SA, using the standard DEdit facilities.

— **SetLimit** : sets the limit of the search tree.

— **Mode** : chose the mode of demonstration.

— **Shortform** : it enables or disabled the control for occurrence, so it is not possible to unify the variables already bound in that piece of unification to themselves.

— **Trace** : it enables in a separate window the tracing of demonstration.

— **Trace PM** : it enables the tracing of the unifier.

— **LoadHKB** : loads a H Knowledge base in the environment: the name of KB is shown beside the window; do not provide the .HKB extension to the name: the system does it.

— **SaveHKB** : saves the current environment in a KB: don't provide extension.

— **EraseEnv** : erases the entire environment.

— **Exit** : exits and closes window.

All the windows that H uses ("Show window", "Trace window", "PM Trace window") has the middle button capability in order to allow to dribble into a file everything that is printed in the window; it is very similar to CHAT's dribble option.

THE H DELIVER

There are a lot of functions available from Lisp Executive that allow the programmer to use the HKBs previously created:

(H.erase) [Function]

Erases all environment (i.e., predicates and SAs) previously loaded.

(H.load *database*) [Function]

Loads H database into environment.

(H.save *database*) [Function]

Saves all current environment into a file

(H.? *pred1 ... predN*) [NLAMBDA Function]

Start the demonstration of the predicates specified. Return the list of predicates proved with the variablesof the call set .

(H.all *variables conjs*) [Function]

Returns the list of all specified variables that satisfy the predicate(s). Remember that variables must begin with a ':' (semicolon).

(H.any *howmany variables conj*) [Function]

Returns *howmany* instantiation values of variables that make true the predicate(s).

(H.attach *foo lambda–expression*) [NLAMBDA Function]

Defines a SA named *foo* to be the value of the LAMBDA-expression written as the second argument.

(H.addaxiom *axioms–list*) [Function]

Adds new axioms to the existing one for that predicate.

(H.axiom *axioms–list*) [Function]

Defines new axioms for the predicate. Deletes the previous ones.

(H.del *axiom*) [Function]

Deletes a single axiom from the database; the other axioms for that predicates are not touched.

(H.show) [NLAMBDA Function]

Shows the definition of the given predicates.

(H.the *variable conjs*) [Function]

Returns only one value for the variable that satisfies the goal.

(SET.H.MODE mode num) [Function]

Set the mode of demonstration: mode may be one of atoms 'first, 'all, 'interactive. If atom 'limit, then you must provide the number of depth (default 200).

H PRIMITIVES

Both environments have three important primitives SA:

(set *var expr*)

Sets in the current level of unification the variable to the value of the expression expr.

(assert *axiom*)

Assert in the database the given axiom. without erasing the old ones.

(delete *axiom*)

Delete in the database the given axiom.

In the system also is present the cut facility ('/'), that has similar behaviour as PROLOG cut ('!').

H DEMOS

An example of axioms may be this:

```
(((append () :a :a))
 ((append (:a . :b) :c  (:a . :d)) < (append :b :c :d)))
```

You can call from H-executive:

```
 (append (1 2 3) (4 5 6) :d)
```

that returns

```
 ((append (1 2 3) (4 5 6)  (1 2 3 4 5 6))
```

Also, if you have in your database:

```
(((A 1))
 ((A 2)))
```

and

```
(((B 3))
 ((B 4)))
```

you can call from TTY exec:

```
(H.? (A :l))  --> (((A 1)))
```

or

```
(H.? (A :k) (B :o))  -->  (((A 1) (B 3)))
```

Moreover:

```
(H.all ':k '((A :k)))  --> (2 1)
(H.all '(:a :b) '((A :a) (B :b)))  --> ((2 4) (2 3) (1 4) (1 3))
(H.any 2 ':j '((A :j)))  --> (2 1)
```

For demo. load the HKBs H-MAZE,H- BLOCKS and try the following:

```
(showworld)
```

that shows you the block world situation: try then

```
(please (put the red cube on the blue one))
 (please (pick up cube2))
```

and so on

The other examples solve for you the maze problem and other interesting things: discover by yourself how they work ...

REFERENCES

Chester D., Using HCPRVR, Department of Computer Sciences, University of Texas, Austin, June 1980

Chester D, HCPRVR: a logic program interpreter in Lisp, proc AAAI, Department of Computer Sciences, University of Texas, Austin, June 1980

Simmons R.F.,Computaions from the English, Prentice Hall, New Jersey, 1984