HQL’s language specification

H. Hernán Moraldo
http://www.hhm.com.ar/
mg@moraldo.com.ar
June 2005

1 About HQL

HQL stands for ‘HHM’s Quantified Lambda’ and it’s a small programming language I created in 2004, for an Advanced Functional Programming class I had with professor Pablo E. Martínez Lopez. It’s a very simple functional language, whose expressions can involve the use of quantifier operators (for all, exists, etc).

The following document presents the language formal specification, that is, the type inference for the language, and its semantics. It’s the copy of the original hand-written specification, so it might be incomplete.

Visit http://www.hhm.com.ar/ to download the interpreter (with no parser) source code, and for any questions and comments, send mail to mg@moraldo.com.ar

2 A sample program

We want to have an example of the use of HQL, that shows what kind of language it is, and what kind of programs can be written with it.

Next code, shows a program written in HQL’s abstract syntax (as there is no parser yet, you might have to convert it to Haskell data before executing it with the interpreter).

\[ \text{program} = \forall x \in \text{Bool} . (\forall y \in \text{Bool} . (\neg (x \lor y)) \iff (x \to y)) \]

To make this program a real HQL program, all \( \forall \) have to be converted to \( \exists \) form, and binary operators like \( \iff \), \( \lor \) and \( \neg \) have to be converted either to BBBOp form, or to applications of functions as defined in the library (library
is a non empty initial environment, as the one used by the interpreter). The full converted program would be:

\[
\text{program} = \forall x \in \text{Bool} \; \forall y \in \text{Bool} . \\
\text{app} (\text{app} \; \text{iif} (\text{app} \; \text{not} (\text{app} \; \text{app} \; \text{or} \; x) \; y)) (\text{app} \; \text{app} \; \text{if} \; x \; y)
\]

Where code like \text{app} \; \text{or} \; x is applying the function contained in variable \text{or} to the expression \text{x}.

The same program is presented in Haskell syntax as follows:

\[
x\text{Term6}=(\text{tForAllB} \; "x"
\;
(\text{tForAllB} \; "y"
\;
(\text{tIif}
\;
(\text{tOr} \; (\text{tNot} \; \text{TVar} \; "x")) \; (\text{TVar} \; "y"))
\;
(\text{tLogicIf} \; \text{TVar} \; "x") \; (\text{TVar} \; "y"))
\;
))
\;
)
\]

Where functions like \text{tForAllB} and \text{tIif} have been previously defined to make programming much faster.

It’s interesting to ask what type has this program. According to the type inference algorithm, it’s a \text{Bool}. The program semantics is in this case \text{True}, as the program is describing a truth that does not depend on any external variables (excepting those of the library).

## 3 HQL’s language specification

### 3.1 Type inference

#### 3.1.1 Definitions:

\[
\text{Type} = \text{Bool} \mid \text{Int} \mid \text{Type} \rightarrow \text{Type}
\]

\[
e := v \mid c_i \mid c_b \mid \text{app} \; e \; e \mid \text{lam} \; t \; v \; e \mid \\
\text{let} \; v = e \; \text{in} \; e \mid \text{explicitLet} \; v = e \; \in \; e \; \text{in} \; e \mid \\
\exists v \in t . e \mid \neg e \mid \text{if} \; e \; e \; e
\]

#### 3.1.2 Base:

\[
\Gamma \vdash v : s
\]
3.1.3 Integers:

\[ \Gamma \vdash c_i : \text{Int} \]

3.1.4 Booleans:

\[ \Gamma \vdash c_b : \text{Bool} \]

3.1.5 Application:

\[
\frac{
\Gamma \vdash e_1 : t_1 \rightarrow t_2 \quad \Gamma \vdash e_2 : t_1
}{
\Gamma \vdash \text{app} e_1 e_2 : t_2
}
\]

3.1.6 Lambda:

\[
\frac{
\Gamma, v : t_1 \vdash e : t_2
}{
\Gamma \vdash \text{lam} v \in t_1, e : t_1 \rightarrow t_2
}\]

3.1.7 Let .. in:

\[
\frac{
\Gamma \vdash e_1 : t_1 \\
\Gamma, v : t_1 \vdash e_2 : t_2
}{
\Gamma \vdash \text{let} v = e_1 \text{ in } e_2 : t_2
}\]

3.1.8 Explicit let .. in

\[
\frac{
\Gamma \vdash e_1 : t \\
\Gamma, v : t \vdash e_2 : t_2
}{
\Gamma \vdash \text{explicitLet} v = e_1 \in t \text{ in } e_2 : t_2
}\]

3.1.9 If .. then .. else:

\[
\frac{
\Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t
}{
\Gamma \vdash \text{ifthenelse} e_1 e_2 e_3 : t
}\]
3.1.10 Exists:

\[\Gamma_v,v : ht \vdash e : \text{Bool} \]
\[\Gamma \vdash \exists v \in t.e : \text{Bool} \]

h would require some extra detail to be thrown in this specification. For now, let’s just say that the t in \(\exists v \in t.e\) is a special type, that can be bool or a restricted integer, so h is a function that converts these special types into the ones accepted by the system.

3.1.11 Not:

\[\Gamma \vdash e : \text{Bool} \]
\[\Gamma \vdash \text{not } e : \text{Bool} \]

3.1.12 IIIOp (others operators are similar):

\[\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \]
\[\Gamma \vdash \text{IIIOp } s e_1 e_2 : \text{Int} \]

3.2 Semantics

3.2.1 Environments:

Environments are sets of triples like this: \((\beta, v, t)\), with v being a variable name, t the expression associated to v (its value), and \(\beta\) the environment where v is defined.

3.2.2 Big K definition:

\[K(\alpha, v, e) = (\alpha + K(\alpha, v, e), v, e)\]

3.2.3 Integer constants:

\[[[c_i]]_\alpha := c_i\]

3.2.4 Boolean constants:

\[[[c_b]]_\alpha := c_b\]
3.2.5 Variables:

$$[[v]]_\alpha := [[t]]_\beta, \text{ if } (\beta, v, t) \in \alpha$$

3.2.6 Lambdas:

$$[[\text{lam } v \in s.e]]_\alpha := \text{lam } v \ e \ \alpha$$

3.2.7 Let .. in:

$$[[\text{let } v \ e_1 e_2]]_\alpha := [[e_2]]_\alpha + K(\alpha, v, e_1)$$

3.2.8 App:

$$[[\text{app } e_1 e_2]]_\alpha := [[e_3]]_\beta + K(\alpha, v, e_2) \text{ if } [[e_1]]_\alpha = \text{lam } v \ e_3 \ \beta$$

3.2.9 If .. then .. else:

$$[[\text{ifthenelse } e_1 e_2 e_3]]_\alpha := \begin{cases} [[e_2]]_\alpha, \text{ if } [[e_1]]_\alpha = \text{true} \\ [[e_3]]_\alpha, \text{ otherwise} \end{cases}$$

3.2.10 Not:

$$[[\text{not } e]]_\alpha := \neg [[e]]_\alpha$$

3.2.11 Exists in booleans:

$$[[\exists v \in \text{Bool}.e]]_\alpha := [e]_\alpha + (\alpha, v, \text{True}) \lor [e]_\alpha + (\alpha, v, \text{False})$$

3.2.12 Exists in Range of Integers:

$$[[\exists v \in [i_1..i_2].e]]_\alpha := \begin{cases} [e]_\alpha + (\alpha, v, i_1) \lor [\exists v \in [i_1+1..i_2].e]_\alpha, \text{ if } e_1 \leq e_2 \\ \text{False}, \text{ otherwise} \end{cases}$$
3.2.13 IIIOp:

\[
[[IIIOp \ id \ n_1 \ n_2]]_{\alpha} := \begin{cases} 
[[n_1]]_{\alpha} + [[n_2]]_{\alpha}, & \text{if } id = '+' \\
[[n_1]]_{\alpha} * [[n_2]]_{\alpha}, & \text{if } id = '*' \\
et cetera
\end{cases}
\]

4 Bibliography