Hoapata Logic Programs are Monotonic

Marc.Bezem@ii.uib.no
Department of Computer Science, University of Bergen, Norway

1 Introduction

Higher-order logic programming [3] is an extension of logic programming where variables for predicates (and predicates of predicates, and so on) are allowed and predicates can be used in terms. The best known example of a higher-order logic programming language is Miller's λ-Prolog, see [2]. Even in a simple setting of [1], without λ-abstraction, higher-order logic programming is not extensional, in contrast to first-order logic programming. Extensionality means, very roughly, that the semantics of a logic program can be explained in terms of the set-theoretic extensions of the relations involved. This allows one to reason about the program by ordinary extensional logic. (Of course non-extensional, or intensional, programming systems have their merits, in particular a greater expressivity. The relevant trade-off here is expressivity versus semantical simplicity.)

The source of the non-extensionality of higher-order logic programs are the syntactic equality tests in the unification procedure. In [1] we identified the class of hoapata programs and proved them to be extensional. The terminology 'hoapata' refers to the higher-order arguments (hoa), which can only be applied (a), passed (p) and/or thrown away (ta) in such programs. Hoapata programs are characterized by a simple syntactic criterion: the higher-order arguments in the head of any program clause should all be distinct variables. Thus we obtained a decidable, syntactic criterion which is sufficient for extensionality. The criterion is not necessary, and cannot be so given the undecidability of extensionality in general.

There exist very simple examples of extensional programs that are not hoapata. Consider the following program P (the clauses are followed by some typings):

```
p(a).                              % p: D->A
p_named(p).                        % p_named: (D->A)->A
```

Here the second clause is not hoapata, since `p` is a higher-order argument which is a constant. (`D` and `A` are the base types for individuals and propositions, respectively.) Nevertheless the program is extensional, since there is no other unary predicate extensionally equal to `p`. (Adding a clause `q(a)` would indeed violate the extensionality.)

The point we want to clear up in this note is the following. Given the signature with constants `a,p,p_named` typed as in the program above, there is a hoapata program P' *having the same initial model semantics* as P:

```
p(a).                                   % p: D->A
p_named(P).                             % p_named: (D->A)->A
```

Here `P` is a variable of type `D->A`, following the Prolog convention to denote variables as capitals. This raises the question whether there exists an extensional program which is not semantically equivalent (in the sense of the initial model semantics) to any hoapata program. The answer is yes, and an example is the following program P'':

```
p(a).                                   % p: D->A
q(a).                                   % q: D->A
q(b).
p_named(p).                             % p_named: (D->A)->A
```

This program is extensional since `p` and `q` have different extensions. It is not hoapata because of the last clause. Obviously, changing `p_named(p)` into `p_named(P)` or `p_named(P) :- P(a)` would also make `p_named(q)` true and thus changes the semantics. Finally, there is no hoapata program which is semantically equivalent to P'', since hoapata programs are *monotonic*. This structural property of hoapata programs will be made precise and proved in the next section. Let us apply it here already in an informal way to P''. Obviously, the extension of `p` is included in the extension of `q`. Since `p_name` is true of `p`, monotonicity would require `p_name` also to be true of `q`, and this is not the case for P''.

2 Monotonicity

For unexplained notations and terminology we refer to [1, 3].

Definition 2.1 Let P be a higher-order logic program. By induction on the type τ we define quasi-orderings $\leq_\tau$ on the set of closed terms of type τ. We put $\leq_D$ to be equality on D. In the other base case we put $t \leq_A t'$ iff $t \in M_P$ implies $t' \in M_P$. For higher types the quasi-orderings are defined pointwise: $t \leq_{\tau \to \tau'} t'$ iff $ts \leq_{\tau'} t's$ for all closed $s : \tau$. We call P monotonic if $ts \leq_{\tau'} ts'$ for all closed $t : \tau \to \tau'$ and $s \leq_\tau s'$.

Obviously, if P is extensional, then we can collapse the quasi-orderings $\leq_\tau$ modulo extensional equality into partial orderings. The program P'' above shows that extensional programs need not be monotonic. However:

Theorem 2.2 *Any hoapata program is monotonic.*

Proof: Let P be a hoapata program and $\vec{z}$ a sequence of distinct higher-order variables. Let $P(n)$ be the following proposition: for every $t : A$ headed by a constant and with free variables among $\vec{z}$, for all closed substitutions σ, σ' such that $\vec{z}^{\sigma} \leq \vec{z}^{\sigma'}$ holds componentwise, if $t^{\sigma} \in T_P{\uparrow}n$, then $t^{\sigma'} \in M_P$. By induction we prove $P(n)$ for all n, which suffices for monotonicity The base case $n = 0$ holds vacuously, since $T_P{\uparrow}0 = \emptyset$. For the induction case, assume $P(n)$ and let conditions be as above, with $t^{\sigma} \in T_P{\uparrow}(n{+}1)$. Now [1, Lemma 4] applies, stating essentially that all higher-order arguments that do play a role in a derivation are eventually applied. Hence there is a (possibly empty) conjunction G of atoms, all starting with a head variable from $\vec{z}$, such that M_P is closed under $t \leftarrow G$ and $G^{\sigma} \subseteq T_P{\uparrow}n$. We want to infer $t^{\sigma'} \in M_P$, which by the above property of G reduces to $G^{\sigma'} \subseteq M_P$. For each $z_i\vec{t'} \in G$ we apply $P(n)$ to $z_i^{\sigma}\vec{t'}$ to get that $z_i^{\sigma}\vec{t'}^{\sigma'}$ is in M_P, and hence by $z_i^{\sigma} \leq z_i^{\sigma'}$ also $z_i^{\sigma'}\vec{t'}^{\sigma'} \in M_P$. This proves $P(n+1)$ and completes the proof of Theorem 2.2.

Acknowledgement

The author is indebted to Dag Normann who conjectured the monotonicity property.

References

[1] M.A. Bezem. An improved extensionality criterion for higher-order logic programs. In L. Fribourg, editor, *Proceedings CSL'01*, LNCS 2142, pp. 203–216. Springer-Verlag, Berlin, September 2001.

[2] D.A. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of logic and computation*, 2(4):497–536, 1991.

[3] G. Nadathur and D.A. Miller. Higher-order logic programming. In D. Gabbay e.a. (eds.) *Handbook of logic in artificial intelligence*, Vol. 5, pp. 499–590. Clarendon Press, Oxford, 1998.