Huet-Lang Second-Order Matching Algorithm

J Strother Moore
Department of Computer Sciences
University of Texas at Austin
Austin, Tx 78712
moore@cs.utexas.edu (512) 471-9590

December 7, 2003

1 Sources

This is my description of Huet and Lang's second-order matching algorithm. This description was based on that in "Tractable and Intractable Second-Order Matching Problems," by Hirata, Yamada, and Harso, Department of Artificial Intelligence, Kyushu Institute of Technology, Kawazu 680-4, Iizuka 820-8502, Japan, as retrieved from http://citeseer.nj.nec.com/cache/papers/cs/14818/-http:zSzzSzdumbo.ai.kyutech.ac.jpzSzhiratazSzpaperszSzCOCOON99.pdf/hirata99tractable.pdf. The algorithm is attributed to

```
@article(Huet78,
 author="G. Huet and B. Lang",
 title="Proving and applying program transformations expressed with
   second-order patterns",
 journal = "Acta Informatica",
 volume="11",
 pages="31--55",
 year="1997")
```

While second-order *unification* is undecidable, second-order *matching* is NP-complete. The Hirata-Yamada-Harso paper shows that some restricted cases of the matching problem are solvable in polynomial time and other restricted cases are still NP-complete. Here, I am interested in the unrestricted case, at least for ACL2 terms.

2 Definitions

A second order term is an ACL2 term in which some of the function symbols are constrained functions. For the purposes of exposition, I will denote constrained function symbols and individual variable symbols in lower case and defined (or axiomatized primitives) in upper case.

Thus, `(F (G X) Y)` is a term, `(F (G (X)) (Y))` is a first-order ground term, and `(f u v)` is a second-order term.

We assume no t_i or s_i contains a lambda expression. But we will introduced (and eliminate) lambda expressions below as we develop the algorithm.

The challenge is to find a substitution σ such that t_i/σ is identical to s_i, for every i.

A substitution is a finite map from constrained function symbols and individual variable symbols. Individual variable symbols are mapped to first-order terms. Constrained function symbols of arity

n are mapped to lambda expressions of the form (lambda $(v_1 ... v_n)$ β) where β is a term whose only free variables are among the v_i.

Thus, { u := `(G (X))`; v := `(Y)` ; f := `(LAMBDA (i j) (F i j))` } is a substitution.

The *beta reduction* of `((LAMBDA` $(v_1 \ldots v_n)$ β`)` $\alpha_1 \ldots \alpha_n$`)` is $\beta/\{v_1 := \alpha_1; \ldots v_n := \alpha_n\}$.

Thus, the beta reduction of `((LAMBDA (i j) (F i j)) (G (X)) (Y))` is `(F (G (X)) (Y))`.

The result, $t/\sigma = t'$, of applying a subsitution σ to a (second-order) term t is obtained as follows.

- If t is a variable symbol in the domain of σ, t' is $\sigma(t)$.
- If $t =$ `(`$fa_1 \ldots a_k$`)` where f is in the domain of σ, then t' is the beta reduction of `(`$\sigma(f)a_1/\sigma \ldots a_n/\sigma$`)`
- If t is a variable symbol not in the domain of σ, or t is a constant, t' is t.
- Otherwise, t is of the form `(`$fa_1 \ldots a_k$`)` and t' is `(`$fa_1/\sigma \ldots a_n/\sigma$`)`.

A *matching expression* E is a finite set of ordered pairs, $\langle t_i, s_i \rangle$ such that t_i is a second order term and s_i is a first order ground term.

A matching expression E is said to be *matchable* iff there exists a substitution σ such that t_i/σ is s_i, for all $\langle t_i, s_i \rangle$ in E.

The *second-order matching problem* is the problem of determining whether a given E is matchable.

3 The Algorithm

The Huet-Lang algorithm for the second-order matching problem is based on three transformation rules that map matching expressions to matching expressions. We give the three rules below. We say $E \Rightarrow E'$ is there is a transformation that maps E to E'. We say $E \Rightarrow^* E'$ if there is a finite sequence of E_i such that $E \Rightarrow E_1 \Rightarrow E_2 \ldots \Rightarrow E'$.

The *Huet-Lang transformation rules* are the last three rules shown below. The others are my own addition and correct either a trivial ommission in my sources or a misunderstanding of them by me.

- **Identity**:
  $\{\langle \mathtt{s}, \mathtt{s} \rangle\} \cup E \Rightarrow E$.
- **Binding**:
  $\{\langle \mathtt{v}, \mathtt{s} \rangle\} \cup E \Rightarrow E/\{\mathtt{v} := \mathtt{s}\}$, where `v` is a variable symbol.
- **Simplification**:
  $\{\langle$`(F`$t_1 \ldots t_n$`)`, `(F`$s_1 \ldots s_n$`)`$\rangle\} \cup E \Rightarrow \{\langle t_1, s_1 \rangle, \ldots \langle t_n, s_n \rangle\} \cup E$.
- **Projection**:
  $E \Rightarrow E/\{f :=$ `(LAMBDA (V1 ... Vn) Vi)`$\}$, if one of the elements of E is $\langle$`(f` $t_1 \ldots t_n$`)`$, s\rangle$.
- **Imitation**:
  $E \Rightarrow E/\{f :=$ `(LAMBDA (V1 ... Vn) (F (h1 V1 ... Vn) ... (h`m `V1 ... Vn)))`$\}$, if one of the elements of E is $\langle$`(f` $t_1 \ldots t_n$`)`, `(F` $s_1 \ldots s_m$`)`$\rangle$ and the `h`m are new "constrained" function symbols.

Huet-Lang Theorem: E is matchable iff $E \Rightarrow \phi$.

The first three rules imply the usual recursive descent through terms with matching concrete function symbols, with bindings created when necessary. Otherwise, we have only the case in which the function symbol, `f`, of the pattern is a variable. Projection implies that we must consider – among the other possibilities – that `f` returns one of its arguments and we must try, in turn, to match each argument with the target. Imitation implies that if the target is a (necessarily concrete) function call, we make the body of `f` supply that call and we set up new matching problems for each argument position.

The rules are non-deterministic, so we must search. We must generate new function symbols as we go. The substitution of the `LAMBDA` expressions for `f` result in beta reduction, so the locals of the `LAMBDA`s need never be explicit.