Type Inference For Recursive Definitions

Assaf J. Kfoury
Church Project
Boston University
kfoury@cs.bu.edu

Santiago M. Pericás-G.
Church Project
Boston University
santiago@cs.bu.edu

June 30, 1999
Type Inference for Recursive Definitions

- What we studied ... 
  - *Type inference with universal types, recursive types and object types.*

- Our approach ... 
  - *Based on a stratification by rank (Leivant).*

- What we included ... 
  - *Recursive definitions in a broader sense:*
    - *Recursive functions via fix.*
    - *Objects a la Abadi and Cardelli.*
Example 1: Polymorphic Stacks

Typable at rank-1 of our system:

\[
\text{stack} \triangleq \\
[\text{isempty} = \text{true}, \\
\text{top} = \varsigma(s)s.\text{top}, \\
\text{pop} = \varsigma(s)s, \\
\text{push} = \varsigma(s)\lambda x.((s.\text{pop} := s).\text{isempty} := \text{false}).\text{top} := x]
\]

\[
: \forall t_1.\mu t_2.[\text{isempty} : \text{bool}, \text{top} : t_1, \text{pop} : t_2, \text{push} : t_1 \rightarrow t_2]
\]

**Typability at rank-1 of our system = poly-time decidable :-)**
Example 2: Transposition of a Matrix

Typable at rank-2 of our system (but not in ML):

```
let val map1 = map
    fun map2 f ([]) = []
    | map2 f ([]::_) = []
    | map2 f (lst) = (f hd lst)::map2 f (f tl lst)
in
    map2 map1 [[1,2],[3,4]] (* [[1,3],[2,4]] *)
end

: ∀t₁. (∀t₂. (list(t₁) → t₂) → list(list(t₁))) → list(list(t₁)) → list(list(t₁))
```

Typability at rank-2 of our system = exp-time decidable :-)

Example 3: Transposition of a List of Matrices

Typable at rank-3 of our system:

```ml
let fun map3 f g [] = []
    | map3 f g lst = (f g (hd lst))::map3 f g (tl lst)

in
    map3 map2 map1 [ [[1,2],[3,4]],
                     [[5,6,7],[8,9,10]] ]

end
```

*Typability at rank-3 of our system = undecidable :-(*
Summary of Results

:-) Typability at our rank-1 system $\Lambda_1^{\text{fix}}$ is decidable: Algorithm TI.

:-) Typability at our rank-2 system $\Lambda_2^{\text{fix}}$ is decidable: Via reduction to typability at rank-1.

:-( Undecidability of Regular Semi-Unification, i.e. finding regular solutions to instances of (unrestricted) semi-unification.

⇒ Typability at rank-1 with polymorphic recursion (system $\Lambda_1^{\text{fix}}$) is undecidable.

⇒ Typability at our rank-3 system $\Lambda_3$ is undecidable.

⇒ Typability at our rank-$k$ system $\Lambda_k$ for $k > 3$ is undecidable.
Definitions

- **Language:**
  \[ M, N ::= c \mid x \mid \lambda x. M \mid MN \mid \text{fix } x. M \mid [\ell_i = \varsigma(x)M_i^{i \in I}] \mid M.\ell \mid \]
  \[ M.\ell \leftarrow \varsigma(x)N \]

- **Types:**
  \[ T_0^{\text{Ob}} = \text{TVar} \cup Q \cup \{(\sigma \rightarrow \tau) \mid \sigma, \tau \in T_0^{\text{Ob}}\} \cup \{ (\mu t.\sigma) \mid \sigma \in T_0^{\text{Ob}} \} \]
  \[ \quad \cup \{ [\ell_i : \tau_i^{i \in I}] \mid \tau_i \in T_0^{\text{Ob}} \} \]
  \[ T_k^{\text{Ob}} = T_k^{\text{Ob}} \cup \{(\sigma \rightarrow \tau) \mid \sigma \in T_k^{\text{Ob}}, \tau \in T_{k+1}^{\text{Ob}}\} \cup \{ (\forall t.\sigma) \mid \sigma \in T_{k+1}^{\text{Ob}} \} \]
  \[ T_{k+1}^{\text{Ob},-} = T_k^{\text{Ob}} \cup \{(\sigma \rightarrow \tau) \mid \sigma \in T_k^{\text{Ob}}, \tau \in T_{k+1}^{\text{Ob},-}\} \]

- **New rule for typing recursive functions:**
  \[ (\text{Monofix}) \quad \frac{E \cup \{ x : \sigma \} \vdash M : \sigma}{E \vdash \text{fix } x. M : \sigma} \quad \sigma \in T_k^{-} \]
System $\Lambda_1^{\text{fix-}}$

- Algorithm TI: online, bottom-up, unification-based and polynomial-time.
- Example: $M \equiv [x = 1, \text{getx} = \varsigma(s)s.x, \text{gets} = \varsigma(s)s]$

<table>
<thead>
<tr>
<th>Subterm</th>
<th>Environment</th>
<th>Type</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>int</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$s$</td>
<td>${s : t_1}$</td>
<td>$t_1$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$s.x$</td>
<td>${s : t_1}$</td>
<td>$t_3$</td>
<td>${t_1 \leq [x : t_3]}$</td>
</tr>
<tr>
<td>$s$</td>
<td>${s : t_2}$</td>
<td>$t_2$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$M$</td>
<td>$\emptyset$</td>
<td>$S(\mu t_1.[x : \text{int}, \text{getx} : t_3, \text{gets} : t_1])$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$S = \text{Unify(\text{Equate(}\{\mu t_1.[x : \text{int}, \text{getx} : t_3 : \text{gets} : t_1] \leq [x : t_3]\})\})$

$= \text{Unify(}\{[x : \text{int}] = [x : t_3]\})$

$= \{(t_3, \text{int})\}$
**System $\Lambda_2^{\text{fix}}$**

- Typability in $\Lambda_2^{\text{fix}}$ is equivalent to finding regular solutions to instances of *Acyclic Semi-Unification*.

- Type inference algorithm based on a reduction to type inference at rank-1, i.e. system $\Lambda_1^{\text{fix}}$. Complexity: DEXPTIME-complete.

- Rank-2 rule for typing recursive definitions. The type $\sigma$ is of the form $(\forall \overrightarrow{t_1} \cdots) \rightarrow (\forall \overrightarrow{t_2} \cdots) \rightarrow \cdots \rightarrow \tau$ where $\tau$ is quantifier free.

\[
\begin{array}{c}
\text{(Monofix)} \\
E \cup \{ x : \sigma \} \vdash M : \sigma \\
\hline
E \vdash \text{fix } x. M : \sigma \\
\end{array}
\] 

$\sigma \in T_2^{\text{fix}}$
**System $\Lambda_1^{\text{fix}}$**

- **Word Problem (by example):** Given the monoid $(\{a, b\}^*, \cdot)$ and the set of constraints $E = \{aaa = a, aab = bb\}$, determine if $abb$ represents the same element as $aaaaaabb$ according to $E$,

$$aaaaaabb = (aaa)aab = aaab = a(aab) = abb$$

**The word problem over monoids is undecidable.**

- **Regular Semi-Unification (by example):** Given the set $\{t_1 \rightarrow \text{bool} \leq t_2, t_3 \rightarrow \text{int} \leq (t_3 \rightarrow t_1) \rightarrow \text{int}\}$, we need to find $S$ such that there exist $S_1$ and $S_2$ where,

$$S_1(S(t_1 \rightarrow \text{bool})) =_\mu S(t_2) \quad S_2(S(t_3 \rightarrow \text{int})) =_\mu S((t_3 \rightarrow t_1) \rightarrow \text{int})$$

Solution: $S = \{(t_2, t_4 \rightarrow \text{bool}), (t_3, \mu t_3. t_3 \rightarrow t_1)\}$, $S_1 = \{(t_1, t_4)\}$ and $S_2 = \emptyset$. 
System $\Lambda_1^{\text{fix}}$

- For every monoid $M$, set of constraints $E$ and equation $e$ we can construct an instance of semi-unification $\Gamma$ such that the word problem is solvable for $M$ with respect to $E$ and $e$ iff $\Gamma$ has a regular solution.

The regular semi-unification problem is undecidable.

- For every instance $\Gamma$ of semi-unification we can construct a term $M_\Gamma$ such that $\Gamma$ has a regular solution iff $M_\Gamma$ is typable in $\Lambda_1^{\text{fix}}$.

Type inference in $\Lambda_1^{\text{fix}}$ is undecidable.
System $\Lambda_3$

- Let $\sigma \triangleq (\forall t.\text{list}(t) \rightarrow \text{int}) \in T_1$ and $\text{type(FIX)} = \forall t.(t \rightarrow t) \rightarrow t$,

\[
\begin{align*}
\text{(fix } f.\lambda x.\text{if null } x \text{ then } 0 \text{ else } 1 + f(\text{tl } x)) \quad &\sigma \in T_1 \\
\text{f } = \lambda x.\text{if null } x \text{ then } 0 \text{ else } 1 + f(\text{tl } x) \quad &\sigma \in T_1 \\
\text{F } = \lambda f.\lambda x.\text{if null } x \text{ then } 0 \text{ else } 1 + f(\text{tl } x) \quad &\sigma \rightarrow \sigma \in T_2 \\
\text{(FIX F') } \quad &\sigma \in T_1
\end{align*}
\]

The type of FIX must be instantiated to $(\sigma \rightarrow \sigma) \rightarrow \sigma \in T_3$ in the derivation that types (FIX F').

- A term $M$ is typable in $\Lambda_{\text{fix}}^1$ iff $\theta(M)$ is typable in $\Lambda_3$.

**Type inference in $\Lambda_3$ is undecidable.**
System $\Lambda_k$

- Define,

$$\varphi(M) = \begin{cases} 
  x & \text{if } M = x \\
  c & \text{if } M = c \\
  (\lambda x. \varphi(N)) & \text{if } M = (\lambda x. N) \\
  ((\lambda z.z)\varphi(N)\varphi(P)) & \text{if } M = (NP) 
\end{cases}$$

- A term $M$ is typable in $\Lambda_k$ iff $\varphi(M)$ is typable in $\Lambda_{k+1}$.

*Type inference in $\Lambda_k$ for $k \geq 3$ is undecidable.*
Future Work ...

- Analyze typability in systems where ∀'s (only at the top) are combined with ∧'s. Advantages:
  - Principal typings at every finite rank.
  - Typability decidable at every finite rank.
- Investigate conditions under which subtyping (possibly restricted) can be added to our system without turning type inference undecidable.
- Consider type systems in which quantifiers, recursive types and object types can be mixed freely.
More Info ...

- Technical report:
  - http://www.cs.bu.edu/students/grads/santiago/Papers

- Sigma: an interpreter that implements the rank-1 type inference algorithm described in the technical report.