# Type Inference for Bimorphic Recursion

Makoto Tatsuta (National Institute of Informatics)
joint work with:
Ferruccio Damiani (Torino University)

Seminar

School of Computer Science and Engineering, Seoul National University
March 15, 2011

# Introduction

Bimorphic recursion

- one type is for its recursive calls in the body of its definition
- the other type is for its calls outside its definition
- nested

Results:
(1) Type inference for bimorphic recursion
(2) Undecidability of type inference without instantiation

Ideas:

- Local type inference for recursion
- Reduction to semiunification problems

# Bimorphic Recursion Type System BR

Expressions $e ::= x|c|\lambda x.e|ee|\mathsf{rec}\{x = e\}$

Types $u, v, w ::= \alpha|\mathsf{bool}|\mathsf{int}|u \to u|u \times u|u$ list

Substitution $s$ function from type variables to types
- $\{\alpha|s(\alpha) \neq \alpha\}$ finite
- $\mathsf{Dom}(s) = \{\alpha|s(\alpha) \neq \alpha\}$

Type environment $U ::= \{x_1 : u_1, \ldots, x_n : u_n\}$ where $x_i \neq x_j$ for $i \neq j$
Judgment $U \vdash e : u$.

Inference rules

$$\frac{}{U, x : u \vdash x : u}\,(ass) \qquad \frac{}{U \vdash c : s(u)}\,(con) \quad (\mathsf{type}(c) = u)$$

$$\frac{U, x : u \vdash e : v}{U \vdash \lambda x.e : u \to v}\,(\to I) \qquad \frac{U \vdash e_1 : v \to u \quad U \vdash e_2 : v}{U \vdash e_1 e_2 : u}\,(\to E)$$

$$\frac{U, x : s_1(u) \vdash e : u}{U \vdash \mathsf{rec}\{x = e\} : s_2(u)}\,(rec) \quad (\mathsf{Dom}(s_1), \mathsf{Dom}(s_2) \subseteq \mathsf{FTV}(u) - \mathsf{FTV}(U))$$

# Pairs and Numbers

Pairs and projections:

$\text{type}(\text{pair}) = \alpha_1 \to \alpha_2 \to (\alpha_1 \times \alpha_2)$
$\text{type}(\text{fst}) = (\alpha_1 \times \alpha_2) \to \alpha_1$
$\text{type}(\text{snd}) = (\alpha_1 \times \alpha_2) \to \alpha_2$

Abbreviations:

$(e_1, e_2) = \text{pair } e_1 e_2$
$e.1 = \text{fst } e$
$e.2 = \text{snd } e$

Natural numbers:

$\text{type}(0) = \text{int},\ \text{type}(1) = \text{int},\ \text{type}(2) = \text{int},\ \ldots$
$\text{type}(\text{pred}) = \text{int} \to \text{int}$
$\text{type}(\text{d}) = \text{int} \to \alpha \to \alpha$

Abbreviations:

$n - 1 = \text{pred}(n)$
$\text{if } n = 0 \text{ then } e_1 \text{ else } e_2 = \text{d} n e_1 e_2$

# Example 1

$\mathsf{Rev} = \lambda x.(\mathsf{Rev2}\ (\lambda y.y))x,$
$\mathsf{Rev2} = \mathsf{rec}\{f_2 = \lambda zw.\mathsf{if}\ (\mathsf{null}\ w)\ \mathsf{then}\ z[\,]$
$\quad\quad \mathsf{else}\ f_2(\lambda xy.z(yx))(\mathsf{tl}\ w)(\lambda x.[(\mathsf{hd}\ w).x])\}$
$\mathsf{Rev}\ [0,1,2] = [2,1,0]$
$(\mathsf{Rev2}\ d)l = d(\mathsf{Rev}\ l)$

$d$ a dispatcher that returns $f_n(\ldots(f_1 x)\ldots)$ when it takes $x$ and continuations $f_1, \ldots, f_n$

- Not typable in ML (monomorphic recursion)
- Typable in BR

$e_2 = \lambda zw.\mathsf{if}\ (\mathsf{null}\ w)\ \mathsf{then}\ z[\,]\ \mathsf{else}\ f_2(\lambda xy.z(yx))(\mathsf{tl}\ w)(\lambda x.[(\mathsf{hd}\ w).x])$
$f_2 : (\beta\ \mathsf{list} \to (\beta\ \mathsf{list} \to \beta\ \mathsf{list}) \to \alpha) \to \beta\ \mathsf{list} \to (\beta\ \mathsf{list} \to \beta\ \mathsf{list}) \to \alpha$
$\quad \vdash e_2 : (\beta\ \mathsf{list} \to \alpha) \to \beta\ \mathsf{list} \to \alpha.$

by $(rec)$ with $s_1(\alpha) = (\beta\ \mathsf{list} \to \beta\ \mathsf{list}) \to \alpha$ and $s_2(\alpha) = \beta\ \mathsf{list}$

$\vdash \mathsf{Rev2} : (\beta\ \mathsf{list} \to \beta\ \mathsf{list}) \to \beta\ \mathsf{list} \to \beta\ \mathsf{list}$
$\vdash \mathsf{Rev} : \beta\ \mathsf{list} \to \beta\ \mathsf{list}$

# Example 2

$e_0 = [0, 1, 2],$
$e_3 = \lambda zw.\text{if (null } w\text{) then } (\lambda x.z[\,])(f_4(\lambda x.x)e_0)$
$\qquad \text{else } f_3(\lambda xy.z(yx))(\text{tl } w)(\lambda x.[(\text{hd } w).x])$
$\text{Rev3} = e_3[f_3 := \text{Rev3}, f_4 := \text{Rev4}],$
$\text{Rev4} = \text{Rev3}.$

Rev3, Rev4 the same as Rev2 except reversing the fixed list $e_0$ as a dummy task

$\text{Rev4} = \text{rec}\{f_4 = \text{rec}\{f_3 = e_3\}\},$
$\text{Rev3} = \text{rec}\{f_3 = e_3[f_4 := \text{Rev4}]\}.$

They are bimorphic recursion

# Example 2 (cont.)

$$\begin{aligned}&f_4 : (\text{int list} \to \text{int list}) \to \text{int list} \to \text{int list},\\ &f_3 : (\beta\ \text{list} \to (\beta\ \text{list} \to \beta\ \text{list}) \to \alpha) \to \beta\ \text{list} \to (\beta\ \text{list} \to \beta\ \text{list}) \to \alpha\\ &\qquad \vdash e_3 : (\beta\ \text{list} \to \alpha) \to \beta\ \text{list} \to \alpha.\end{aligned}$$

By $(rec)$ rule, we have

$$\begin{aligned}&f_4 : (\text{int list} \to \text{int list}) \to \text{int list} \to \text{int list}\\ &\qquad \vdash \text{rec}\{f_3 = e_3\} : (\beta\ \text{list} \to \beta\ \text{list}) \to (\beta\ \text{list} \to \beta\ \text{list}).\end{aligned}$$

By $(rec)$, we have

$$\vdash \text{Rev4} : (\text{int list} \to \text{int list}) \to \text{int list} \to \text{int list}.$$

We also have

$$\begin{aligned}&f_3 : (\beta\ \text{list} \to (\beta\ \text{list} \to \beta\ \text{list}) \to \alpha) \to \beta\ \text{list} \to (\beta\ \text{list} \to \beta\ \text{list}) \to \alpha\\ &\qquad \vdash e_3[f_4 := \text{Rev4}] : (\beta\ \text{list} \to \alpha) \to \beta\ \text{list} \to \alpha\end{aligned}$$

and by $(rec)$ we have

$$\vdash \text{Rev3} : (\beta\ \text{list} \to \beta\ \text{list}) \to \beta\ \text{list} \to \beta\ \text{list}.$$

## Related Work

[Henglein 89]
- Non-nested bimorphic recursion
- Decidable type inference

[Comini et al 08]
- Equivalence between abstract interpretation Gori and Levi, and bimorphic recursion type system

## Main Theorem

**Theorem (Type inference).** (1) There is a type inference algorithm for the type system BR. That is, there is an algorithm such that for a given term it returns its principal type if the term is typable, and it returns the fail if the term is not typable.
(2) The typability in the type system BR is decidable. That is, there is an algorithm that decides if there is some $u$ such that $\vdash e : u$ for a given $e$.

# Type Inference and Semiunification

A type inference problem is reduced to a semiunification problem

a semiunification problem with a single inequation  decidable

a semiunification problem with two inequations  undecidable

$\{M \leq N\}$ has solution $r(s(M)) = s(N)$ for some $r, s$

$\{M_1 \leq N_1, M_2 \leq N_2\}$ has solution $r_1(s(M_1)) = s(N_1), r_2(s(M_2)) = s(N_2)$ for some $r_1, r_2, s$

A known algorithm reduces the bimorphic type inference to a semiunification problem with two inequations

Ours reduces the bimorphic type inference to a semiunification problem with a single inequation

# Local Type Inference

As type inference algorithms for simply typed lambda calculus and let polymorphism, when a term $e$, a type $u$, and a type environment $U$ are given, the algorithm chases the proof of $U \vdash e : u$ upward from the conclusion, and produces a set of equations between types such that the existence of its unifier $s$ is equivalent to the provability of $s(U) \vdash e : s(u)$. Then we had difficulty for the $(rec)$ rule.

The idea is that we follow the above algorithm but we handle the the $(rec)$ rule in a separate way. First we choose an uppermost $(rec)$ rule in the proof:

$$\dfrac{\begin{array}{c}\vdots\,\pi_1 \\ U, x : s_1(u) \vdash e : u\end{array}}{\begin{array}{c}U \vdash \mathrm{rec}\{x = e\} : s_2(u) \\ \vdots\,\pi_2\end{array}}\,(rec)$$

Let $V$ be $\mathrm{FTV}(u) - \mathrm{FTV}(U)$. We will use $\pi_3$ to denote the subproof with the $(rec)$ rule and $\pi_1$. The subproof $\pi_2$ cannot access to $V$ because $s_2$ hides $V$ and $U$ does not have any information of $V$.

Hence type inference for $\pi_3$ can be done separately from $\pi_2$. Since $\pi_3$ has only one $(rec)$ rule, the type inference for $\pi_3$ is reduced to a semiunification problem with a single inequation. Hence type inference for $\pi_3$ is possible since there is an algorithm solving a semiunification problem with a single inequation. By this, we will have a most general semiunifier $s$ and a principal type $v$ of $\mathrm{rec}\{x = e\}$. Then our type inference is reduced to type inference of the proof $\pi_4$:

$$\dfrac{}{s(U) \vdash \mathrm{rec}\{x = e\} : s_1(v)}\,(axiom)$$
$$\vdots\, s(\pi_2)$$

for some $s_1$ where $s(\pi_2)$ denotes the proof obtained from $\pi_2$ by replacing every judgment $U_1 \vdash e_1 : u_1$ by $s(U_1) \vdash e_1 : s(u_1)$ and the rule $(axiom)$ denotes a temporary axiom. Since this reduction eliminates one $(rec)$ rule, by repeating this reduction, we can reduce our type inference problem to type inference problem for some term without the $(rec)$ rule. Hence we can complete type inference by solving it with the type inference algorithm for the simply typed lambda calculus.

## Algorithm

Algorithm $E$

- input typing problem
- outputs a pair of a unification problem and a substitution
- $E(U \vdash e : u) = (E_0, s_0)$
- the typing problem is transformed to a unification problem $E_0$ and a partial solution $s_0$

$E(U, x : u \vdash x : v) = (\{u = v\}, 1)$,

$E(U \vdash x : u) = (\{\text{bool} = \text{int}\}, 1)$ where $x \notin \text{Dom}(U)$,

$E(U \vdash c : u) = (\{u = v\}, 1)$ where $\text{type}(c) = v$,

$E(U \vdash \lambda x.e_1 : u) = (E_1 \cup \{s_1(\alpha \to \beta) = s_1(u)\}, s_1)$ where

  $\alpha, \beta$ fresh type variables,

  $E(U, x : \alpha \vdash e_1 : \beta) = (E_1, s_1)$,

$E(U \vdash e_1 e_2 : u) = (s_2(E_1) \cup E_2, s_2 s_1)$ where

  $\alpha$ a fresh type variable,

  $E(U \vdash e_1 : \alpha \to u) = (E_1, s_1)$,

  $E(s_1(U) \vdash e_2 : s_1(\alpha)) = (E_2, s_2)$,

$E(U \vdash \text{rec}\{x = e_1\} : u) = (\{s_2 s_1(u) = s_2 s_1(\alpha)\}, s_2 s_1)$ where

  $\alpha, \beta$ fresh type variables,

  $E(U, x : \beta \vdash e_1 : \alpha) = (E_1, s_1)$,

  $U = \{x_1 : u_1, \ldots, x_n : u_n\}$,

  $\vec{u} = u_1 \times \ldots \times u_n$,

  $s_2 = \text{mgu}(E_1 \cup \{s_1(\alpha \times \vec{u}) \leq s_1(\beta \times \vec{u})\})$,

$E(U \vdash \text{rec}\{x = e_1\} : u) = (\{\text{bool} = \text{int}\}, 1)$ where

  $\alpha, \beta$ fresh type variables,

  $E(U, x : \beta \vdash e_1 : \alpha) = (E_1, s_1)$,

  $U = \{x_1 : u_1, \ldots, x_n : u_n\}$,

  $\vec{u} = u_1 \times \ldots \times u_n$,

  $\text{mgu}(E_1 \cup \{s_1(\alpha \times \vec{u}) \leq s_1(\beta \times \vec{u})\}) = \text{fail}$.

## Correctness Theorem

**Theorem (Correctness).**
(1) If $E(U \vdash e : u) = (E_0, s_0)$ and $s$ is a unifier of $E_0$, then $ss_0$ is a solution of the typing problem $U \vdash e : u$.
(2) If $E(U \vdash e : u) = (E_0, s_0)$, the typing problem $U \vdash e : u$ has a solution $s$, $V$ is a finite set of type variables, and $\mathsf{FTV}(U) \cup \mathsf{FTV}(u) \subseteq V$, then there is a unifier $s_0'$ of $E_0$ such that $s_0' s_0 =_V s$.

This theorem can be extended to a type system with let polymorphism

# Bimorphic Recursion with No Instantiation

Type system BRNI for bimorphic recursion with no instantiation

Replacing the rule $(rec)$

$$\frac{U, x : s_1(u) \vdash e : u}{U \vdash \mathsf{rec}\{x = e\} : s_2(u)}\,(rec) \quad (\mathsf{Dom}(s_1), \mathsf{Dom}(s_2) \subseteq \mathsf{FTV}(u) - \mathsf{FTV}(U))$$

by the rule $(recni)$

$$\frac{U, x : s_1(u) \vdash e : u}{U \vdash \mathsf{rec}\{x = e\} : u}\,(recni) \quad (\mathsf{Dom}(s_1) \subseteq \mathsf{FTV}(u) - \mathsf{FTV}(U))$$

**Theorem (Undecidability).** The typability in BRNI is undecidable

Reasons:

**Instantiation Property.**
In BR, if $U \vdash e : u$ is provable, then $s(U) \vdash e : s(u)$ is provable.

Instantiation property fails in BRNI

## Semiunification

Semiunification terms $M, N ::= \alpha | M \times M$

where $\alpha$ is a type variable

**Theorem (Henglein89)** The existence of a semiunifier of the set of two inequations is undecidable. That is, there is no algorithm that decides if there is some $s$ such that $s_1(s(M_1)) = s(N_1)$ and $s_2(s(M_2)) = s(N_2)$ for some $s_1, s_2$ for a given semiunification problem $\{M_1 \leq N_1, M_2 \leq N_2\}$.

**Theorem (Henglein89)** Type inference for polymorphic recursion is undecidable

The proof reduced type inference for polymorphic recursion to semiunification problems

We will reduce type inference for BRNI to semiunification problems by refining Henglein's proof

# Proof Sketch of Undecidability

$K = \lambda xy.x$
$(e_1 \doteq e_2) = \lambda y.(ye_1, ye_2) \qquad (y \notin \mathsf{FTV}(e_1 e_2))$

The expression $\widetilde{M}$ is defined for a semiunification term $M$ by
- $\widetilde{\alpha}_i = z_i$
- $\widetilde{M_1 \times M_2} = (\widetilde{M}_1, \widetilde{M}_2)$

**Lemma.** Let $\vec{\alpha} = \mathsf{FTV}(M_1, M_2, N_1, N_2)$ and $\vec{z} = \widetilde{\vec{\alpha}}$. Let

$e_1 = \mathsf{rec}\{f = \lambda\vec{z}.K(\widetilde{M}_1, \widetilde{M}_2)(\lambda\vec{y}.(f\vec{y}.1 \doteq \widetilde{N}_1))\}$
$e_2 = \mathsf{rec}\{f = \lambda\vec{z}.K(\widetilde{M}_1, \widetilde{M}_2)(\lambda\vec{y}.(f\vec{y}.2 \doteq \widetilde{N}_2))\}$

where $\vec{y}$ are fresh variables of the same length as $\vec{z}$

The judgment $\vdash e_1 \doteq e_2 : u$ is provable in BRNI for some $u$ if and only if the semiunification problem $\{M_1 \leq N_1, M_2 \leq N_2\}$ has a semiunifier

# Conclusion

Bimorphic recursion

- one type is for its recursive calls in the body of its definition
- the other type is for its calls outside its definition
- nested

Results:

(1) Type inference for bimorphic recursion

(2) Undecidability of type inference without instantiation

Ideas:

- Local type inference for recursion
- Reduction to semiunification problems