Typage

Further issues (and a solution)

2018-2019

Giovanni Bernardi, `gioXYZirif.fr`
`http://www.irif.fr/~gio/index.xhtml`
Université Paris Diderot

Questions questions questions . . .

1. What is a partial order / poset?

Questions questions questions . . .

1. What is a partial order / poset?
2. When is a function over a poset *monotone* ?

Questions questions questions . . .

1. What is a partial order / poset?
2. When is a function over a poset *monotone* ?
3. What is a complete lattice ?

Questions questions questions ...

1. What is a partial order / poset?
2. When is a function over a poset *monotone* ?
3. What is a complete lattice ?
4. What is a fixed-point of a function ?

Questions questions questions . . .

1. What is a partial order / poset?
2. When is a function over a poset *monotone* ?
3. What is a complete lattice ?
4. What is a fixed-point of a function ?
5. What does the Knaster-Tarski theorem state ?

Plan

1. History
2. Reality check motivation
3. Types as open terms
4. Semantic equivalence
5. Types as graphs
6. Unification for graphs

Subtyping and recursive types are common in modern programming languages. For example, Java [14] [. . .] allows interfaces to be mutually recursive, although there is no unfolding rule. [. . .] What is not common is type inference for real languages with subtyping and recursive types.

– *T. Jim, J. Palsberg,*
Type inference in systems of recursive types with subtyping, 1999

Typing $\lambda x.xx$

desiderata: type derivation

$$\dfrac{\dfrac{}{x : A \to y \vdash x : A \to y} \qquad \dfrac{\dfrac{}{x : A \to y \vdash x : A \to y}}{x : A \to y \vdash x : A}\; A \approx A \to y}{\dfrac{x : A \to y \vdash xx : y}{\vdash \lambda x.xx : (A \to y) \to y}}$$

Typing $\lambda x.xx$

desiderata: type derivation

$$\dfrac{\dfrac{}{x : A \to y \vdash x : A \to y} \qquad \dfrac{\dfrac{}{x : A \to y \vdash x : A \to y}}{x : A \to y \vdash x : A}\ A \approx A \to y}{\dfrac{x : A \to y \vdash xx : y}{\vdash \lambda x.xx : (A \to y) \to y}}$$

inference of constraints

$$\dfrac{\dfrac{}{x : t_1 \vdash x : t_3 \to t_2}\ t_1 = t_3 \to t_2 \qquad \dfrac{}{x : t_1 \vdash x : t_3}\ t_1 = t_3}{\dfrac{x : t_1 \vdash xx : t_2}{\vdash \lambda x.xx : t}\ t = t_1 \to t_2}$$

underdetermined system of equations

$$\begin{array}{lcl} t_1 & \stackrel{?}{=} & t_3 \\ t_1 & \stackrel{?}{=} & t_3 \to t_2 \\ t & \stackrel{?}{=} & t_1 \to t_2 \end{array}$$

Type expressions

$$\sigma, \tau \quad ::= \quad t \mid int \mid \mu t.\sigma \mid \sigma \to \sigma \quad \text{where} \quad t \in Vars$$

- $\mu x.\sigma$ binds x in σ, free and bound variables as expected
- σ *contractive* if for any subexpression of σ of the form $\mu x.\mu t_1.\mu t_2.\ldots.\mu t_n.\tau$, the term τ is not x
- T_μ set of contractive terms

when are two type expressions equal ?

$$\boxed{\begin{array}{lcl} \mu t.(t \to t') \to t' & \stackrel{?}{=} & \mu t.t \to t' \\ \mu t.(int \to t) & \stackrel{?}{=} & int \times \mu t.(int \to t) \\ \mu t.t \to t & \stackrel{?}{=} & (\mu t.t \to t) \to (\mu t.t \to t) \end{array}}$$

Type equivalence semantic approach

Let
$\Sigma = Vars \cup \{\rightarrow, int\}$ ranked alphabet
$arity(int) = arity(t) = 0$
$arity(\rightarrow) = 2$

Tree over Σ is

Type equivalence semantic approach

Let
$\Sigma = Vars \cup \{\rightarrow, int\}$ ranked alphabet
$arity(int) = arity(t) = 0$
$arity(\rightarrow) = 2$

Tree over Σ is a partial function $f : \mathbb{N}_+^\star \rightarrow \Sigma$ such that

Type equivalence semantic approach

Let
$\Sigma = \mathit{Vars} \cup \{\rightarrow, \mathit{int}\}$ ranked alphabet
$\mathit{arity}(\mathit{int}) = \mathit{arity}(t) = 0$
$\mathit{arity}(\rightarrow) = 2$

Tree over Σ is a partial function $f : \mathbb{N}_+^\star \rightarrow \Sigma$ such that
$\mathit{dom}(f)$ is a *tree-domain*:
(a) $\mathit{dom}(f)$ non-empty, (b) $\mathit{dom}(f)$ prefix-closed, (c) for all $\pi \in \mathit{dom}(f)$

- $i, j \in N_+^\star, 1 \leq i \leq j$ and $\pi j \in \mathit{dom}(f)$ imply $\pi i \in \mathit{dom}(f)$
- $f(\pi) = A$ of arity $k \geq 0$ implies for $i \in \mathbb{N}_+, \pi i \in \mathit{dom}(f)$ iff $1 \leq i \leq k$

see Section 1.2, Courcelle 1983; Definition 21.2.1 TALP

Type equivalence semantic approach

from [Cardone and Coppo, '91]

Let

- T_R be set of regular trees over $\hat{\Sigma}$
- $\mathit{treeof}(-) : T_\mu \to T_R$ be defined inductively by

$$\begin{array}{lcll} \mathit{treeof}(t) & = & t & t \in \mathit{Vars} \\ \mathit{treeof}(\mathit{int}) & = & \mathit{int} & \\ \mathit{treeof}(\sigma \to \tau) & = & \mathit{treeof}(\sigma) \to \mathit{treeof}(\tau) & \\ \mathit{treeof}(\mu t.\sigma) & = & \mu F & \end{array}$$

where

- $F \triangleq \lambda z.(\mathit{treeof}(\sigma)\{^z/_t\}) : T_R \to T_R$
- μF exists — Theorem 4.10.1, Courcelle '83

Let $\sigma \stackrel{ext}{=} \tau$ whenever $\mathit{treeof}(\sigma) \stackrel{ext}{=} \mathit{treeof}(\tau)$

Type equivalence semantic approach

from [Cardone and Coppo, '91]

Let

- T_R be set of regular trees over $\hat{\Sigma}$
- $\mathit{treeof}(-) : T_\mu \to T_R$ be defined inductively by

things are getting complicated

$$\mathit{treeof}(\mu t.\sigma) \quad = \quad \mu F$$

where

- $F \triangleq \lambda z.(\mathit{treeof}(\sigma)\{^z/_t\}) : T_R \to T_R$
- μF exists Theorem 4.10.1, Courcelle '83

$$\text{Let } \sigma \stackrel{ext}{=} \tau \text{ whenever } \mathit{treeof}(\sigma) \stackrel{ext}{=} \mathit{treeof}(\tau)$$

Types as graphs

$$\sigma, \tau \quad ::= \quad t \mid \mathit{int} \mid \mu t.\sigma \mid \sigma \to \sigma \qquad \text{where } t \in \mathit{Vars}$$

$$\Sigma \quad = \quad \mathit{Vars} \cup \{\to, \mathit{int}\} \qquad \text{type constructors}$$

Graph: (V, E)

- $V \subseteq \Sigma \times \mathit{Id}$ set of nodes
- $E \subseteq V \times V$ set of edges

Types as graphs

$$\sigma, \tau \quad ::= \quad t \mid int \mid \mu t.\sigma \mid \sigma \rightarrow \sigma \qquad \text{where } t \in Vars$$

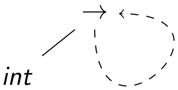
$$\Sigma \quad = \quad Vars \cup \{\rightarrow, int\} \qquad \text{type constructors}$$

Graph: (V, E)

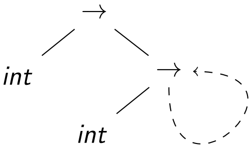
- $V \subseteq \Sigma \times Id$ — set of nodes
- $E \subseteq V \times V$ — set of edges

intuitive mapping

$\mu x.(int \rightarrow x)$



$int \rightarrow \mu x.(int \rightarrow x)$



Unifier

Exercice 7.4 notes par X. Leroy

Let

- (V, E) be a graph
- $cns : V \to \Sigma$ — type constructors in node
- $child : V \to \mathbb{N} \to V$ — i^{th} child of a node

A *substitution* is equivalence relation $R \subseteq V \times V$ such that if $n_1 \; R \; n_2$ and $cns(n_1), cns(n_2) \notin Vars$ then

- $cns(n_1) = cns(n_2)$
- $\forall i \in [1, arity(cns(n_1))]. \, child(i, n_1) \; R \; child(i, n_2)$

A *unifier* for a system of equations E is a substitution R such that $n_1 \; R \; n_2$ for every $n_1 \stackrel{?}{=} n_2 \in E$. A unifier R for a system E is *principal* if for every unifier R' of E, $R' \subseteq R$.

Computing principal unifier

$$
\begin{aligned}
mgu(\emptyset, R) &= R \\
mgu(\{n_1 \stackrel{?}{=} n_2\} \cup E, R) &= mgu(E, R) \; \mathit{if} \; n_1 \; R \; n_2 \\
mgu(\{n_1 \stackrel{?}{=} n_2\} \cup E, R) &= mgu(E, R + \{(n_1, n_2)\}) \\
&\quad \mathit{if} \; cns(n_1) \in Vars \; \mathit{or} \; cns(n_2) \in Vars \\
mgu(\{n_1 \stackrel{?}{=} n_2\} \cup E, R) &= mgu(E \cup \{(child(1, n_1) \stackrel{?}{=} child(1, n_2))\}, \\
&\qquad\qquad \vdots \\
&\qquad\qquad (child(k, n_1) \stackrel{?}{=} child(k, n_2))\}, \\
&\qquad R + \{((n_1, n_2))\}) \\
&\quad \mathit{if} \; cns(n_1) \notin Vars \; \mathit{and} \; cns(n_1) = cns(n_2) \\
mgu(\{n_1 \stackrel{?}{=} n_2\} \cup E, R) &= \text{Nothing} \\
&\quad \mathit{if} \; cns(n_1) \notin Vars \; \mathit{and} \; cns(n_1) \neq cns(n_2)
\end{aligned}
$$

where

$R + \{(n_1, n_2)\}$ smallest equivalence relation containing R and $\{(n_1, n_2)\}$;
and $k = arity(cns(n_1))$

Problem

$\boxed{\text{How to transform syntactic types into graphs ??}}$

$\boxed{\text{How to transform graphs into syntactic types??}}$

Material

- Section 7.4 notes cours “Typage et programmation”, X. Leroy
- Type Inference with Recursive Types: Syntax and Semantics, F. Cardone, M. Coppo, 1991