Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC
http://courses.engr.illinois.edu/cs421

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Type Inference Algorithm

Let infer $(\Gamma, e, \tau) = \sigma$

- Γ is a typing environment (giving polymorphic types to expression variables)
- e is an expression
- τ is a type (with type variables),
- σ is a substitution of types for type variables
- Idea: σ is the constraints on type variables necessary for Γ |- $e : \tau$
- Should have $\sigma(\Gamma)$ |- $e : \sigma(\tau)$

Type Inference Algorithm

infer (Γ, exp, τ) =

- Case *exp* of
  - Var *v* --> return Unify{$\tau \equiv$ freshInstance($\Gamma(v)$)}
    - Replace all quantified type vars by fresh ones
  - Const *c* --> return Unify{$\tau \equiv$ freshInstance φ } where Γ |- $c : \varphi$ by the constant rules
  - fun *x* -> *e* -->
    - Let α, β be fresh variables
    - Let σ = infer ([x: α] + Γ, *e*, β)
    - Return Unify({$\sigma(\tau) \equiv \sigma(\alpha \to \beta)$}) o σ

Type Inference Algorithm (cont)

- Case *exp* of
  - App (e_1 e_2) -->
    - Let α be a fresh variable
    - Let σ_1 = infer(Γ, e_1, $\alpha \to \tau$)
    - Let σ_2 = infer($\sigma_1(\Gamma)$, e_2, $\sigma_1(\alpha)$)
    - Return σ_2 o σ_1

Type Inference Algorithm (cont)

- Case *exp* of
  - If e_1 then e_2 else e_3 -->
    - Let σ_1 = infer(Γ, e_1, bool)
    - Let σ_2 = infer($\sigma_1\Gamma$, e_2, $\sigma_1(\tau)$)
    - Let σ_3 = infer(σ_2 o $\sigma_1(\Gamma)$,e_2,σ_2 o $\sigma(\tau)$)
    - Return σ_3 o σ_2 o σ_1

Type Inference Algorithm (cont)

- Case *exp* of
  - let *x* = e_1 in e_2 -->
    - Let α be a fresh variable
    - Let σ_1 = infer(Γ, e_1, α)
    - Let σ_2 = infer([x:GEN($\sigma_1(\Gamma)$, $\sigma_1(\alpha)$)] + $\sigma_1(\Gamma)$, e_2, $\sigma_1(\tau)$)
    - Return σ_2 o σ_1

Type Inference Algorithm (cont)

- Case *exp* of
  - let rec $x = e_1$ in e_2 -->
    - Let α be a fresh variable
    - Let $\sigma_1 = \text{infer}([x{:}\ \alpha] + \Gamma, e_1, \alpha)$
    - Let $\sigma_2 = \text{infer}([x{:}\text{GEN}(\sigma_1(\Gamma),\sigma_1(\alpha))]$
      $+ \sigma_1(\Gamma), e_2, \sigma_1(\tau))$
    - Return $\sigma_2 \circ \sigma_1$

Type Inference Algorithm (cont)

- To infer a type, introduce type_of
- Let α be a fresh variable
- type_of (Γ, e) =
  - Let $\sigma = \text{infer}(\Gamma, e, \alpha)$
  - Return $\sigma(\alpha)$
- Need an algorithm for Unif

Background for Unification

- Terms made from constructors and variables (for the simple first order case)
- Constructors may be applied to arguments (other terms) to make new terms
- Variables and constructors with no arguments are base cases
- Constructors applied to different number of arguments (arity) considered different
- Substitution of terms for variables

Simple Implementation Background

```
type term = Variable of string
                    | Const of (string * term list)

let rec subst var_name residue term =
    match term with Variable name ->
          if var_name = name then residue else term
        | Const (c, tys) ->
          Const (c, List.map (subst var_name residue)
                              tys);;
```


Unification Problem

Given a set of pairs of terms ("equations")

$$\{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$$

(the *unification problem*) does there exist a substitution σ (the *unification solution*) of terms for variables such that

$$\sigma(s_i) = \sigma(t_i),$$

for all i = 1, …, n?

Uses for Unification

- Type Inference and type checking
- Pattern matching as in OCAML
  - Can use a simplified version of algorithm
- Logic Programming - Prolog
- Simple parsing

Unification Algorithm

- Let $S = \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$ be a unification problem.
- Case S = { }: Unif(S) = Identity function (i.e., no substitution)
- Case $S = \{(s, t)\} \cup S'$: Four main steps

Unification Algorithm

- Delete: if s = t (they are the same term) then Unif(S) = Unif(S')
- Decompose: if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ (same f, same m!), then $\mathrm{Unif}(S) = \mathrm{Unif}(\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S')$
- Orient: if t = x is a variable, and s is not a variable, $\mathrm{Unif}(S) = \mathrm{Unif}(\{(x,s)\} \cup S')$

Unification Algorithm

- Eliminate: if s = x is a variable, and x does not occur in t (the occurs check), then
  - Let $\varphi = x \mid\!\rightarrow t$
  - Let $\psi = \mathrm{Unif}(\varphi(S'))$
  - $\mathrm{Unif}(S) = \{x \mid\!\rightarrow \psi(t)\} \circ \psi$
    - Note: $\{x \mid\!\rightarrow a\} \circ \{y \mid\!\rightarrow b\} = \{y \mid\!\rightarrow (\{x \mid\!\rightarrow a\}(b))\} \circ \{x \mid\!\rightarrow a\}$ if y not in a

Tricks for Efficient Unification

- Don't return substitution, rather do it incrementally
- Make substitution be constant time
  - Requires implementation of terms to use mutable structures (or possibly lazy structures)
  - We won't discuss these

Example

- x,y,z variables, f,g constructors
- S = {(f(x), f(g(y,z))), (g(y,f(y)), x)}

Example

- x,y,z variables, f,g constructors
- S is nonempty
- S = {(f(x), f(g(y,z))), (g(y,f(y)), x)}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (g(y,f(y)), x)

- S = {(f(x), f(g(y,z))), (g(y,f(y)), x)}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (g(y,f(y))), x)
- Orient: (x, g(y,f(y)))
- S = {(f(x), f(g(y,z))), (g(y,f(y)), x)}
- -> {(f(x), f(g(y,z))), (x, g(y,f(y)))}

Example

- x,y,z variables, f,g constructors

- S -> {(f(x), f(g(y,z))), (x, g(y,f(y)))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(x), f(g(y,z)))

- S -> {(f(x), f(g(y,z))), (x, g(y,f(y)))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(x), f(g(y,z)))
- Decompose: (x, g(y,z))
- S -> {(f(x), f(g(y,z))), (x, g(y,f(y)))}
- -> {(x, g(y,z)), (x, g(y,f(y)))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (x, g(y,f(y)))
- Substitute: {x |-> g(y,f(y))}
- S -> {(x, g(y,z)), (x, g(y,f(y)))}
- -> {(g(y,f(y)), g(y,z))}

- With {x |-> g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (g(y,f(y)), g(y,z))
- S -> {(g(y,f(y)), g(y,z))}

With {x |→ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (g(y,f(y)), g(y,z))
- Decompose: (y, y) and (f(y), z)
- S -> {(g(y,f(y)), g(y,z))}
- -> {(y, y), (f(y), z)}

With {x |→ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (y, y)
- S -> {(y, y), (f(y), z)}

With {x |→ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (y, y)
- Delete
- S -> {(y, y), (f(y), z)}
- -> {(f(y), z)}

With {x |→ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(y), z)
- S -> {(f(y), z)}

With {x |→ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(y), z)
- Orient: (z, f(y))
- S -> {(f(y), z)}
- -> {(z, f(y))}

With {x |→ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (z, f(y))
- S -> {(z, f(y))}

With {x |→ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (z, f(y))
- Eliminate: {z|-> f(y)}
- S -> {(z, f(y))}
- -> { }

With {x |→ {z |→ f(y)} (g(y,f(y))) }
o {z |→ f(y)}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (z, f(y))
- Eliminate: {z|-> f(y)}
- S -> {(z, f(y))}
- -> { }

With {x |→ g(y,f(y))} o {(z |→ f(y))}

Example

S = {(f(x), f(g(y,z))), (g(y,f(y)),x)}
Solved by {x |→ g(y,f(y))} o {(z |→ f(y))}

$$f(\underbrace{g(y,f(y))}_{x}) = f(g(y,\underbrace{f(y)}_{z}))$$

and

$$g(y,f(y)) = \underbrace{g(y,f(y))}_{x}$$

Example of Failure: Decompose

- S = {(f(x,g(y)), f(h(y),x))}
- Decompose: (f(x,g(y)), f(h(y),x))
- S -> {(x,h(y)), (g(y),x)}
- Orient: (g(y),x)
- S -> {(x,h(y)), (x,g(y))}
- Eliminate: (x,h(y))
- S -> {(h(y), g(y))} with {x |→ h(y)}
- No rule to apply! Decompose fails!

Example of Failure: Occurs Check

- S = {(f(x,g(x)), f(h(x),x))}
- Decompose: (f(x,g(x)), f(h(x),x))
- S -> {(x,h(x)), (g(x),x)}
- Orient: (g(y),x)
- S -> {(x,h(x)), (x,g(x))}
- No rules apply.