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

Two Problems

- Type checking
  - Question: Does exp. e have type τ in env Γ?
  - Answer: Yes / No
  - Method: Type derivation
- Typability
  - Question Does exp. e have some type in env. Γ? If so, what is it?
  - Answer: Type τ / error
  - Method: Type inference

Type Inference - Outline

- Begin by assigning a type variable as the type of the whole expression
- Decompose the expression into component expressions
- Use typing rules to generate constraints on components and whole
- Recursively find substitution that solves typing judgment of first subcomponent
- Apply substitution to next subcomponent and find substitution solving it; compose with first, etc.
- Apply comp of all substitution to orig. type var. to get answer

Type Inference Algorithm

Let infer (Γ, *e*, τ) = σ

- 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 substitution solving the constraints on type variables necessary for Γ |- *e* : τ
- Should have σ(Γ) |- *e* : σ(τ) valid

Type Inference Algorithm

infer (Γ, *exp*, τ) =

- Case *exp* of
  - Var *v* --> return Unify{τ ≡ freshInstance(Γ(*v*))}
    - Replace all quantified type vars by fresh ones
  - Const *c* --> return Unify{τ ≡ freshInstance φ } where Γ |- *c* : φ by the constant rules
  - fun *x* -> *e* -->
    - Let α, β be fresh variables
    - Let σ = infer ({x: α} + Γ, *e*, β)
    - Return Unify({σ(τ) ≡ σ(α → β)}) ∘ σ

Example of inference with Var Rule

Instance {'a -> 'w} ('w a fresh variable)

{x: All 'a. ('a * 'b) list, y:All. 'b}|- x : (int * string) list

freshInstance(All 'a. ('a * 'b) list) = ('w * 'b) list

Unify {((int*string)list = ('w * 'b) list)} ={'w -> int, 'b -> string}

After substitution:

Instance {'a -> int}

{x:All 'a. ('a * string) list, y:All. string}|- x:(int * string) list

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, $\alpha \rightarrow \tau$)
    - Let σ_2 = infer($\sigma_1(\Gamma)$, e_2, $\sigma_1(\alpha)$)
    - Return $\sigma_2 \circ \sigma_1$

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($\sigma_2 \circ \sigma_1(\Gamma)$,$e_3$,$\sigma_2 \circ \sigma_1(\tau)$)
    - Return $\sigma_3 \circ \sigma_2 \circ \sigma_1$

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 $\sigma_2 \circ \sigma_1$

Type Inference Algorithm (cont)

- Case *exp* of
  - let rec $x = e_1$ in e_2 -->
    - Let α be a fresh variable
    - Let σ_1 = infer({x: α} + 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 $\sigma_2 \circ \sigma_1$

Type Inference Algorithm (cont)

- To infer a type, introduce type_of
- Let α be a fresh variable
- type_of (Γ, e) =
  - Let σ = infer (Γ, e, α)
  - Return σ (α)
- 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 x = Variable "'a";;   let tm = Const ("2",[]);;

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 $i = 1, \ldots, n$?

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

Unification Algorithm

- Eliminate: if $s = x$ is a variable, and x does not occur in t (the occurs check), then
  - Let $\varphi = \{x \to t\}$
    - $\text{Unif}(S) = \text{Unif}(\varphi(S')) \circ \{x \to t\}$
  - Let $\psi = \text{Unif}(\varphi(S'))$
  - $\text{Unif}(S) = \{x \to \psi(t)\} \circ \psi$
    - Note: $\{x \to a\} \circ \{y \to b\} = \{y \to (\{x \to a\}(b))\} \circ \{x \to a\}$ 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
- Unify {(f(x) = f(g(f(z),y))), (g(y,y) = x)} = ?

Example

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

Example

- x,y,z variables, f,g constructors
- Pick a pair: (g(y,y) = x)
- Unify {(f(x) = f(g(f(z),y))), (g(y,y) = x)} = ?

Example

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

by Orient

Example

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

Example

- x,y,z variables, f,g constructors
- {(f(x) = f(g(f(z),y))), (x = g(y,y))} is non-empty

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

Example

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

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

Example

- x,y,z variables, f,g constructors
- Pick a pair: (x = g(y,y))
- Eliminate x with substitution {x→ g(y,y)}
  - Check: x not in g(y,y)
- Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (x = g(y,y))
- Eliminate x with substitution {x→ g(y,y)}

- Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} =
  Unify {(f(g(y,y)) = f(g(f(z),y)))}
  o {x→ g(y,y)}

Example

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

- Unify {(f(g(y,y)) = f(g(f(z),y)))}
  o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- {(f(g(y,y)) = f(g(f(z),y)))} is non-empty

- Unify {(f(g(y,y)) = f(g(f(z),y)))}
  o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(g(y,y)) = f(g(f(z),y)))
- Unify {(f(g(y,y)) = f(g(f(z),y)))} o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(g(y,y)) = f(g(f(z),y)))
- Decompose:(f(g(y,y)) = f(g(f(z),y))) becomes {(g(y,y) = g(f(z),y))}
- Unify {(f(g(y,y)) = f(g(f(z),y)))} o {x→ g(y,y)} = Unify {(g(y,y) = g(f(z),y))} o {x→ g(y,y)}

Example

- x,y,z variables, f,g constructors
- {(g(y,y) = g(f(z),y))} is non-empty
- Unify {(g(y,y) = g(f(z),y))} o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (g(y,y) = g(f(z),y))
- Unify {(g(y,y) = g(f(z),y))} o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(g(y,y)) = f(g(f(z),y)))
- Decompose: (g(y,y)) = g(f(z),y)) becomes {(y = f(z)); (y = y)}
- Unify {(g(y,y) = g(f(z),y))} o {x→ g(y,y)} = Unify {(y = f(z)); (y = y)} o {x→ g(y,y)}

Example

- x,y,z variables, f,g constructors
- Unify {(y = f(z)); (y = y)} o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- {(y = f(z)); (y = y)} o {x→ g(y,y)} is non-empty
- Unify {(y = f(z)); (y = y)} o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (y = f(z))
- Unify {(y = f(z)); (y = y)} o {x→ g(y,y)} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (y = f(z))
- Eliminate y with {y → f(z)}
- Unify {(y = f(z)); (y = y)} o {x→ g(y,y)} =
  Unify {(f(z) = f(z))}
  o ({y → f(z)} o {x→ g(y,y)})=
  Unify {(f(z) = f(z))}
  o {y → f(z); x→ g(f(z), f(z))}

Example

- x,y,z variables, f,g constructors
- Unify {(f(z) = f(z))}
  o {y → f(z); x→ g(f(z), f(z))} = ?

Example

- x,y,z variables, f,g constructors
- {(f(z) = f(z))} is non-empty
- Unify {(f(z) = f(z))}
  o {y → f(z); x→ g(f(z), f(z))} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(z) = f(z))
- Unify {(f(z) = f(z))}
  o {y → f(z); x→ g(f(z), f(z))} = ?

Example

- x,y,z variables, f,g constructors
- Pick a pair: (f(z) = f(z))
- Delete
- Unify {(f(z) = f(z))}
  o {y → f(z); x→ g(f(z), f(z))} =
  Unify {} o {y → f(z); x→ g(f(z), f(z))}

Example

- x,y,z variables, f,g constructors
- Unify {} o {y → f(z); x→ g(f(z), f(z))} = ?

Example

- x,y,z variables, f,g constructors
- {} is empty
- Unify {} = identity function
- Unify {} o {y → f(z); x→ g(f(z), f(z))} =
  {y → f(z); x→ g(f(z), f(z))}

Example

- Unify {(f(x) = f(g(f(z),y))), (g(y,y) = x)} =
  {y → f(z); x→ g(f(z), f(z))}

f(x) = f(g(f(z), y))
→ f(g(f(z), f(z))) = f(g(f(z), f(z)))

g(y , y) = x
→ g(f(z),f(z)) = g(f(z), f(z))

Example of Failure: Decompose

- Unify{(f(x,g(y)) = f(h(y),x))}
- Decompose: (f(x,g(y)) = f(h(y),x))
- = Unify {(x = h(y)), (g(y) = x)}
- Orient: (g(y) = x)
- = Unify {(x = h(y)), (x = g(y))}
- Eliminate: (x = h(y))
- Unify {(h(y) = g(y))} o {x → h(y)}
- No rule to apply! Decompose fails!

Example of Failure: Occurs Check

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