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

Curry - Howard Isomorphism

- Type Systems are logics; logics are type systems
- Types are propositions; propositions are types
- Terms are proofs; proofs are terms

- Function space arrow corresponds to implication; application corresponds to modus ponens

Curry - Howard Isomorphism

- Modus Ponens

$$\frac{A \Rightarrow B \quad A}{B}$$

- Application

$$\frac{\Gamma \mid\!- \ e_1 : \alpha \rightarrow \beta \quad \Gamma \mid\!- \ e_2 : \alpha}{\Gamma \mid\!- \ (e_1\ e_2) : \beta}$$

Mea Culpa

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only meta-variable in the logic)
- Would need:
  - Object level type variables and some kind of type quantification
  - **let** and **let rec** rules to introduce polymorphism
  - Explicit rule to eliminate (instantiate) polymorphism

Support for Polymorphic Types

- Monomorpic Types (τ):
  - Basic Types: int, bool, float, string, unit, …
  - Type Variables: α, β, γ, δ, ε
  - Compound Types: $\alpha \to \beta$, int * string, bool list, …
- Polymorphic Types:
  - Monomorphic types τ
  - Universally quantified monomorphic types
  - $\forall \alpha_1, \ldots, \alpha_n . \tau$
  - Can think of τ as same as $\forall . \tau$

Example: {} |- 2 = 3 : bool

(=) : All 'a. 'a -> 'a -> bool

Instance: 'a specialized to int

$$\dfrac{\dfrac{}{\{\}\ \vdash\ 2 : \text{int}}\ \text{Const} \qquad \dfrac{}{\{\}\ \vdash\ 3 : \text{int}}\ \text{Const}}{\{\}\ \vdash\ 2 = 3 : \text{bool}}\ \text{BinOp}$$

Polymorphic Example

- Assume additional constants and primitive operators:
- (=) :$\forall\alpha.\ \alpha$ -> α -> bool
- hd :$\forall\alpha.\ \alpha$ list -> α
- tl: $\forall\alpha.\ \alpha$ list -> α list
- is_empty : $\forall\alpha.\ \alpha$ list -> bool
- (::) : $\forall\alpha.\ \alpha$ -> α list -> α list
- [] : $\forall\alpha.\ \alpha$ list

Example FreeVars Calculations

- Vars('a -> (int -> 'b) -> 'a) ={'a , 'b}
- FreeVars (All 'b. 'a -> (int -> 'b) -> 'a) =
- {'a , 'b} – {'b}= {'a}
- FreeVars {x : All 'b. 'a -> (int -> 'b) -> 'a}

  id: All 'c. 'c -> 'c,

  y: All 'c. 'a -> 'b -> 'c} =
- {'a} U {} U {'a, 'b} = {'a, 'b}

Support for Polymorphic Types

- Typing Environment Γ supplies polymorphic types (which will often just be monomorphic) for variables
- Free variables of monomorphic type just type variables that occur in it
  - Write FreeVars(τ)
- Free variables of polymorphic type removes variables that are universally quantified
  - $\text{FreeVars}(\forall\alpha_1, \ldots, \alpha_n . \tau) = \text{FreeVars}(\tau) - \{\alpha_1, \ldots, \alpha_n\}$
- FreeVars(Γ) = all FreeVars of types in range of Γ

Monomorphic to Polymorphic

- Given:
  - type environment Γ
  - monomorphic type τ
  - τ shares type variables with Γ
- Want most polymorphic type for τ that doesn't break sharing type variables with Γ
- $Gen(\tau, \Gamma) = \forall \alpha_1, \ldots , \alpha_n . \tau$ where $\{\alpha_1, \ldots , \alpha_n\} = freeVars(\tau) - freeVars(\Gamma)$

Polymorphic Typing Rules

- A *type judgement* has the form

$$\Gamma \mid\!\!- \text{exp} : \tau$$

  - Γ uses polymorphic types
  - τ still monomorphic
- Most rules stay same (except use more general typing environments)
- Rules that change:
  - Variables
  - Let and Let Rec
  - Allow polymorphic constants, polymorphic operators
- Worth noting functions again

Polymorphic Variables (Identifiers)

Variable axiom:

$$\frac{}{\Gamma \mid\!- \; x : \varphi(\tau)} \quad \text{if } \Gamma(x) = \forall \alpha_1, \ldots, \alpha_n . \tau$$

- Where φ replaces all occurrences of $\alpha_1, \ldots, \alpha_n$ by monotypes $\tau_1, \ldots, \tau_n$
- Note: Monomorphic rule special case:

$$\frac{}{\Gamma \mid\!- \; x : \tau} \quad \text{if } \Gamma(x) = \tau$$

- Constants treated same way

Polymorphic Let and Let Rec

- let rule:

$$\frac{\Gamma \;|\!-\; e_1 : \tau_1 \quad \{x : \mathrm{Gen}(\tau_1,\Gamma)\} + \Gamma \;|\!-\; e_2 : \tau_2}{\Gamma \;|\!-\; (\mathrm{let}\ x = e_1\ \mathrm{in}\ e_2) : \tau_2}$$

- let rec rule:

$$\frac{\{x : \tau_1\} + \Gamma \;|\!-\; e_1 : \tau_1 \quad \{x : \mathrm{Gen}(\tau_1,\Gamma)\} + \Gamma \;|\!-\; e_2 : \tau_2}{\Gamma \;|\!-\; (\mathrm{let\ rec}\ x = e_1\ \mathrm{in}\ e_2) : \tau_2}$$

Fun Rule Stays the Same

- fun rule:

$$\frac{\{x : \tau_1\} + \Gamma \mid\!- \; e : \tau_2}{\Gamma \mid\!- \text{fun } x \text{ -> } e : \tau_1 \rightarrow \tau_2}$$

- Types τ_1, τ_2 monomorphic
- Function argument must always be used at same type in function body

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 - Example

- What type can we give to

  (fun x -> fun f -> f (f x))

- Start with a type variable and then look at the way the term is constructed

Type Inference - Example

- First approximate:

  $\{\ \} \vert\text{- (fun x -> fun f -> f (f x))} : \alpha$

- Second approximate: use fun rule

$$\frac{\{x : \beta\} \mid\!\text{- (fun f -> f (f x))} : \gamma}{\{\ \} \mid\!\text{- (fun x -> fun f -> f (f x))} : \alpha}$$

- Remember constraint $\alpha \equiv (\beta \rightarrow \gamma)$

Type Inference - Example

- Third approximate: use fun rule

$$\frac{\dfrac{\{f : \delta \,;\, x : \beta\} \ |\!-\ f\ (f\ x) : \varepsilon}{\{x : \beta\} \ |\!-\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}}{\{\ \} \ |\!-\ (\text{fun } x \text{ -> fun } f \text{ -> } f(f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma)$; $\gamma \equiv (\delta \rightarrow \varepsilon)$

Type Inference - Example

- Fourth approximate: use app rule

$$\frac{\dfrac{\dfrac{\{f:\delta;\ x:\beta\} \mid\!-\ f : \varphi \rightarrow \varepsilon \quad \{f:\delta;\ x:\beta\} \mid\!-\ f\ x : \varphi}{\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!-\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}}{\{\ \} \mid\!-\ (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma);\ \gamma \equiv (\delta \rightarrow \varepsilon)$

Type Inference - Example

- Fifth approximate: use var rule, get constraint $\delta \equiv \varphi \to \varepsilon$, Solve with same
- Apply to next sub-proof

$$\dfrac{\dfrac{}{\{f:\delta;\ x:\beta\} \mid\!-\ f : \varphi \to \varepsilon} \quad \{f:\delta;\ x:\beta\} \mid\!-\ f\ x : \varphi}{\dfrac{\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon}{\dfrac{\{x : \beta\} \mid\!-\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}{\{\ \} \mid\!-\ (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}}}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$

Type Inference - Example

- Current subst: $\{\delta \equiv \varphi \to \varepsilon\}$

$$\frac{\dfrac{\dfrac{\dots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \to f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \to \text{fun } f \to f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$

Type Inference - Example

- Current subst: $\{\delta \equiv \varphi \to \varepsilon\}$ Use App Rule

$$\frac{\{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!-\ f:\zeta\to\varphi \quad \{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!-\ x:\zeta}{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!-\ f\ x : \varphi}$$

$$\frac{\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon}{\{x : \beta\} \mid\!-\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}$$

$$\{\ \} \mid\!-\ (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$

Type Inference - Example

- Current subst: $\{\delta \equiv \varphi \to \varepsilon\}$
- Var rule: Solve $\zeta \to \varphi \equiv \varphi \to \varepsilon$ Unification

$$\dfrac{\dfrac{\dfrac{\dfrac{\{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!-\ f:\zeta\to\varphi \quad \{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!-\ x:\zeta}{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!-\ f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!-\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}}{\{\ \} \mid\!-\ (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$

Type Inference - Example

- Current subst: $\{\zeta \equiv \varepsilon,\ \varphi \equiv \varepsilon\} \circ \{\delta \equiv \varphi \rightarrow \varepsilon\}$
- Var rule: Solve $\zeta \rightarrow \varphi \equiv \varphi \rightarrow \varepsilon$ Unification

$$\frac{\dfrac{\dfrac{\{f:\varphi\rightarrow\varepsilon;\ x:\beta\} \vdash f:\zeta\rightarrow\varphi \quad \{f:\varphi\rightarrow\varepsilon;\ x:\beta\} \vdash x:\zeta}{\ldots \quad \{f:\varphi \rightarrow \varepsilon;\ x:\beta\} \vdash f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \vdash (f\ (f\ x)) : \varepsilon}}{\dfrac{\{x : \beta\} \vdash (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}{\{\ \} \vdash (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}}$$

- $\alpha \equiv (\beta \rightarrow \gamma);\ \gamma \equiv (\delta \rightarrow \varepsilon)$

Type Inference - Example

- Current subst: $\{\zeta\equiv\varepsilon,\ \varphi\equiv\varepsilon,\ \delta\equiv\varepsilon\to\varepsilon\}$
- Apply to next sub-proof

$$\dfrac{\dfrac{\dots \qquad \{f:\varepsilon\to\varepsilon;\ x:\beta\}\ |\!-\ x:\varepsilon}{\dots \qquad \{f:\varphi\to\varepsilon;\ x:\beta\}\ |\!-\ f\ x : \varphi}}{\dfrac{\dfrac{\{f:\delta\ ;\ x:\beta\}\ |\!-\ (f\ (f\ x)) : \varepsilon}{\{x:\beta\}\ |\!-\ (\mathrm{fun}\ f\ \texttt{->}\ f\ (f\ x)) : \gamma}}{\{\ \}\ |\!-\ (\mathrm{fun}\ x\ \texttt{->}\ \mathrm{fun}\ f\ \texttt{->}\ f\ (f\ x)) : \alpha}}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$

Type Inference - Example

- Current subst: $\{\zeta \equiv \varepsilon,\ \varphi \equiv \varepsilon,\ \delta \equiv \varepsilon \rightarrow \varepsilon\}$
- Var rule: $\varepsilon \equiv \beta$

$$\dfrac{\dots \qquad \dfrac{\dots \qquad \dfrac{}{\{f:\varepsilon \rightarrow \varepsilon;\ x:\beta\} \mid\!-\ x:\varepsilon}}{\{f:\varphi \rightarrow \varepsilon;\ x:\beta\} \mid\!-\ f\ x : \varphi}}{\dfrac{\dfrac{\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon}{\{x : \beta\} \mid\!-\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}}{\{\ \} \mid\!-\ (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}}$$

- $\alpha \equiv (\beta \rightarrow \gamma);\ \gamma \equiv (\delta \rightarrow \varepsilon)$

Type Inference - Example

- Current subst: {ε≡β}o{ζ≡ε, φ≡ε, δ≡ε→ε}
- Solves subproof; return one layer

$$\dfrac{\dots \qquad \dfrac{}{\{f:\varepsilon\to\varepsilon;\ x:\beta\} \mid\!- \ x:\varepsilon}}{\dfrac{\dots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- \ f\ x : \varphi}{\dfrac{\{f : \delta\ ;\ x : \beta\} \mid\!- \ (f\ (f\ x)) : \varepsilon}{\dfrac{\{x : \beta\} \mid\!- \ (\text{fun } f \to f\ (f\ x)) : \gamma}{\{\ \} \mid\!- \ (\text{fun } x \to \text{fun } f \to f\ (f\ x)) : \alpha}}}}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$

Type Inference - Example

- Current subst: $\{\varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \rightarrow \beta\}$
- Solves this subproof; return one layer

$$\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\dots \quad \{f:\varphi \rightarrow \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma);\ \gamma \equiv (\delta \rightarrow \varepsilon)$

Type Inference - Example

- Current subst: $\{\varepsilon\equiv\beta,\ \zeta\equiv\beta,\ \varphi\equiv\beta,\ \delta\equiv\beta\rightarrow\beta\}$
- Need to satisfy constraint $\gamma \equiv (\delta \rightarrow \varepsilon)$, given subst, becomes: $\gamma \equiv ((\beta\rightarrow\beta) \rightarrow \beta)$

$$\dfrac{\dfrac{\dfrac{\dots}{\{f : \delta\ ;\ x : \beta\}\ |\text{-}\ (f\ (f\ x)) : \varepsilon}}{\{x : \beta\}\ |\text{-}\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}}{\{\ \}\ |\text{-}\ (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma)$; $\gamma \equiv (\delta \rightarrow \varepsilon)$

Type Inference - Example

- Current subst:

$\{\gamma \equiv ((\beta \to \beta) \to \beta), \varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \to \beta\}$

- Solves subproof; return one layer

$$\frac{\dfrac{\dfrac{\dots}{\{f : \delta\ ;\ x : \beta\}\ \text{|-}\ (f\ (f\ x)) : \varepsilon}}{\{x : \beta\}\ \text{|-}\ (\text{fun } f \text{ -> } f\ (f\ x)) : \gamma}}{\{\ \}\ \text{|-}\ (\text{fun } x \text{ -> fun } f \text{ -> } f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma); \gamma \equiv (\delta \to \varepsilon)$

Type Inference - Example

- Current subst:

$\{\gamma \equiv ((\beta \rightarrow \beta) \rightarrow \beta), \varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \rightarrow \beta\}$

- Need to satisfy constraint $\alpha \equiv (\beta \rightarrow \gamma)$ given subst: $\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta))$

$$\dfrac{\dfrac{\cdots}{\{x : \beta\} \mid\!- \text{(fun f -> f (f x))} : \gamma}}{\{\ \} \mid\!- \text{(fun x -> fun f -> f (f x))} : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma)$;

Type Inference - Example

- Current subst:

$\{\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta)),$

$\gamma \equiv ((\beta \rightarrow \beta) \rightarrow \beta), \varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \rightarrow \beta\}$

- Solves subproof; return on layer

$$\frac{\overline{\qquad\qquad\qquad}}{\{x : \beta\} \text{ |- (fun f -> f (f x))} : \gamma}$$

$$\{\ \} \text{ |- (fun x -> fun f -> f (f x))} : \alpha$$

Type Inference - Example

- Current subst:

$\{\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta)),$

$\gamma \equiv ((\beta \rightarrow \beta) \rightarrow \beta), \varepsilon \equiv \beta, \zeta \equiv \beta, \varphi \equiv \beta, \delta \equiv \beta \rightarrow \beta\}$

- Done: $\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta))$

{ } |- (fun x -> fun f -> f (f x)) : α

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 $\sigma(\Gamma)$ |- $e : \sigma(\tau)$ valid

Type Inference Algorithm

infer (Γ, *exp*, τ) =

- Case *exp* of
  - Var *v* --> return Unify{$\tau \equiv$ freshInstance(Γ(*v*))}
    - Replace all quantified type vars by fresh ones
  - Const *c* --> return Unify{$\tau \equiv$ 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({$\sigma(\tau) \equiv \sigma(\alpha \rightarrow \beta)$}) $\circ\ \sigma$

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($\sigma_2 o \sigma_1(\Gamma)$,$e_3$,$\sigma_2 o \sigma_1(\tau)$)
    - 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($\{x: \alpha\} + \Gamma, e_1, \alpha$)
    - 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 $Unif(S) = Unif(S')$
- Decompose: if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ (same f, same m!), then
  $Unif(S) = Unif(\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S')$
- Orient: if $t = x$ is a variable, and s is not a variable, $Unif(S) = Unif(\{(x = s)\} \cup S')$

Unification Algorithm

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

- 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 $\{(g(y,y) = g(f(z),y))\}$
  $o\ \{x \rightarrow g(y,y)\}$ = ?

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 $\{y \to f(z)\}$
- Unify $\{(y = f(z)); (y = y)\}$ o $\{x \to g(y,y)\}$ =
  Unify $\{(f(z) = f(z))\}$
    o $(\{y \to f(z)\}$ o $\{x \to g(y,y)\})$=
  Unify $\{(f(z) = f(z))\}$
    o $\{y \to f(z); x \to g(f(z), f(z))\}$

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 $\rightarrow$ 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.