

Programming Languages and Compilers (CS 421)



Elsa L Gunter

2112 SC, UIUC

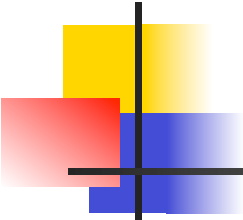
<http://courses.engr.illinois.edu/cs421>

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



Support for Polymorphic Types

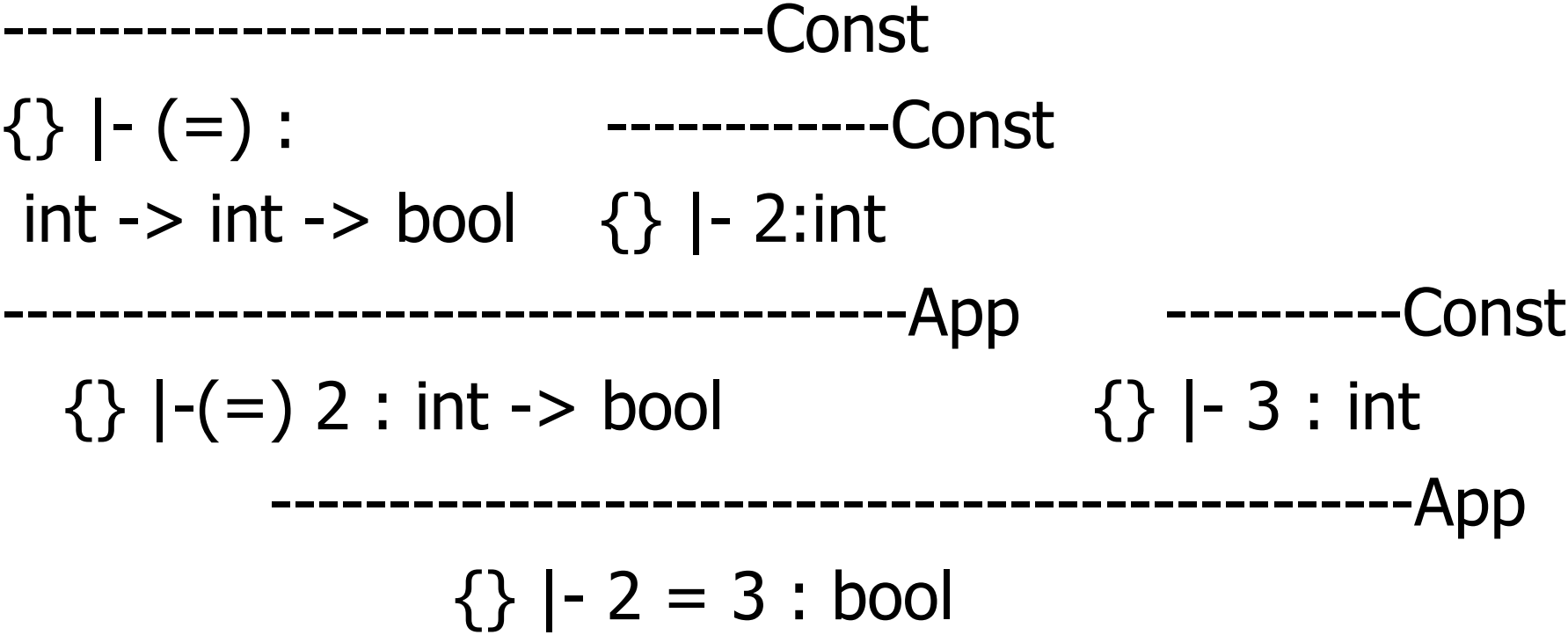
- Monomorphic Types (τ):
 - Basic Types: `int`, `bool`, `float`, `string`, `unit`, ...
 - Type Variables: α , β , γ , δ , ε
 - Compound Types: $\alpha \rightarrow \beta$, `int * string`, `bool list`, ...
- Polymorphic Types:
 - Monomorphic types τ
 - Universally quantified monomorphic types
 - $\forall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



Example: $\{\} \vdash 2 = 3 : \text{bool}$

$(=) : \text{All } 'a. 'a \rightarrow 'a \rightarrow \text{bool}$

Instance: $'a \rightarrow \text{int}$





Example FreeVars Calculations

- $\text{Vars}(\text{'a} \rightarrow (\text{int} \rightarrow \text{'b}) \rightarrow \text{'a}) = \{\text{'a}, \text{'b}\}$
- $\text{FreeVars} (\text{All } \text{'b}. \text{'a} \rightarrow (\text{int} \rightarrow \text{'b}) \rightarrow \text{'a}) =$
- $\{\text{'a}, \text{'b}\} - \{\text{'b}\} = \{\text{'a}\}$
- $\text{FreeVars} \{x : \text{All } \text{'b}. \underline{\text{'a}} \rightarrow (\text{int} \rightarrow \text{'b}) \rightarrow \underline{\text{'a}},$
- $\text{id} : \text{All } \text{'c}. \text{'c} \rightarrow \text{'c},$
- $y : \text{All } \text{'c}. \underline{\text{'a}} \rightarrow \text{'b} \rightarrow \text{'c}\} =$
- $\{\text{'a}\} \cup \{\} \cup \{\text{'a}, \text{'b}\} = \{\text{'a}, \text{'b}\}$



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 $\text{FreeVars}(\tau)$
- Free variables of polymorphic type removes variables that are universally quantified
 - $\text{FreeVars}(\forall \alpha_1, \dots, \alpha_n . \tau) = \text{FreeVars}(\tau) - \{\alpha_1, \dots, \alpha_n\}$
- $\text{FreeVars}(\Gamma) =$ 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
- $\text{Gen}(\tau, \Gamma) = \forall \alpha_1, \dots, \alpha_n . \tau$ where
 $\{\alpha_1, \dots, \alpha_n\} = \text{freeVars}(\tau) - \text{freeVars}(\Gamma)$



Polymorphic Typing Rules

- A *type judgement* has the form
$$\Gamma \vdash \text{exp} : \tau$$
 - 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
- Worth noting functions again



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
 $(\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x))$
- Start with a type variable and then look at the way the term is constructed



Type Inference - Example

- First approximate:

$$\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha$$

- Second approximate: use fun rule

$$\frac{\{x : \beta\} \vdash (\text{fun } f \rightarrow f (f x)) : \gamma}{\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha}$$

$$\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha$$

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



Type Inference - Example

- Third approximate: use fun rule

$$\{f : \delta ; x : \beta\} \vdash f (f x) : \varepsilon$$

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

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



Type Inference - Example

- Fourth approximate: use app rule

$$\frac{\{f:\delta; x:\beta\} \vdash f : \varphi \rightarrow \varepsilon \quad \{f:\delta; x:\beta\} \vdash f x : \varphi}{\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon}$$

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

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

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

Type Inference - Example

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

$$\frac{\{f:\delta; x:\beta\} \vdash f : \varphi \rightarrow \varepsilon \quad \{f:\delta; x:\beta\} \vdash f x : \varphi}{\{f:\delta; x:\beta\} \vdash f (f x) : \varepsilon}$$

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

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

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

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



Type Inference - Example

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

$$\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi$$

$$\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon$$

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

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

Type Inference - Example

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

$$\frac{\{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f:\zeta \rightarrow \varphi \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash x:\zeta}{\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi}$$

$$\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi$$

$$\frac{\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon}{\{x : \beta\} \vdash (\text{fun } f \rightarrow f (f x)) : \gamma}$$

$$\frac{\{x : \beta\} \vdash (\text{fun } f \rightarrow f (f x)) : \gamma}{\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha}$$

$$\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha$$

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

Type Inference - Example

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

$$\frac{\{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f:\zeta \rightarrow \varphi \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash x:\zeta}{\dots}$$

$$\frac{\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi}{\dots}$$

$$\frac{\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon}{\dots}$$

$$\frac{\{x : \beta\} \vdash (\text{fun } f \text{ -> } f (f x)) : \gamma}{\dots}$$

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

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

Type Inference - Example

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

$$\frac{\{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f:\zeta \rightarrow \varphi \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash x:\zeta}{\dots}$$

$$\frac{\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi}{\dots}$$

$$\frac{\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon}{\dots}$$

$$\frac{\{x : \beta\} \vdash (\text{fun } f \text{ -> } f (f x)) : \gamma}{\dots}$$

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

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

Type Inference - Example

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

$$\dots \quad \{f:\varepsilon \rightarrow \varepsilon; x:\beta\} \vdash x:\varepsilon$$

$$\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi$$

$$\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon$$

$$\{x : \beta\} \vdash (\text{fun } f \text{ -> } f (f x)) : \gamma$$

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

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

Type Inference - Example

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

$$\begin{array}{c}
 \dots \quad \frac{\{f:\varepsilon \rightarrow \varepsilon; x:\beta\} \vdash x:\varepsilon}{\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi} \\
 \frac{\{f:\delta; x:\beta\} \vdash (f (f x)) : \varepsilon}{\{x:\beta\} \vdash (\text{fun } f \rightarrow f (f x)) : \gamma} \\
 \{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha
 \end{array}$$

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

Type Inference - Example

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

$$\begin{array}{c}
 \dots \quad \frac{\{f:\varepsilon \rightarrow \varepsilon; x:\beta\} \vdash x:\varepsilon}{\dots \quad \{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi} \\
 \frac{\{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi}{\{f:\delta; x:\beta\} \vdash (f (f x)) : \varepsilon} \\
 \frac{\{f:\delta; x:\beta\} \vdash (f (f x)) : \varepsilon}{\{x:\beta\} \vdash (\text{fun } f \text{ -> } f (f x)) : \gamma} \\
 \frac{\{x:\beta\} \vdash (\text{fun } f \text{ -> } f (f x)) : \gamma}{\{\} \vdash (\text{fun } x \text{ -> } \text{fun } f \text{ -> } f (f x)) : \alpha}
 \end{array}$$

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

Type Inference - Example

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

...

... $\{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi$

$\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon$

$\{x : \beta\} \vdash (\text{fun } f \text{ -> } f (f x)) : \gamma$

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

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



Type Inference - Example

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

...

$$\frac{\frac{\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon}{\{x : \beta\} \vdash (\text{fun } f \rightarrow f (f x)) : \gamma}}{\{\} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha}$$

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

Type Inference - Example

- Current subst:

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

- Solves subproof; return one layer

...

$$\{f : \delta ; x : \beta\} \vdash (f (f x)) : \varepsilon$$

$$\{x : \beta\} \vdash (\text{fun } f \text{ -> } f (f x)) : \gamma$$

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

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

Type Inference - Example

- Current subst:

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

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

...

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

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



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

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



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:** $\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta))$

$\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f (f x)) : \alpha$



Type Inference Algorithm

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

- 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 $\Gamma \vdash e : \tau$
- Should have $\sigma(\Gamma) \vdash e : \sigma(\tau)$ valid



Type Inference Algorithm

$\text{infer}(\Gamma, \text{exp}, \tau) =$

- Case exp of
 - $\text{Var } v \rightarrow$ return $\text{Unify}\{\tau \equiv \text{freshInstance}(\Gamma(v))\}$
 - Replace all quantified type vars by fresh ones
 - $\text{Const } c \rightarrow$ return $\text{Unify}\{\tau \equiv \text{freshInstance } \varphi\}$ where $\Gamma \vdash c : \varphi$ by the constant rules
 - $\text{fun } x \rightarrow e \rightarrow$
 - Let α, β be fresh variables
 - Let $\sigma = \text{infer}(\{x: \alpha\} + \Gamma, e, \beta)$
 - Return $\text{Unify}(\{\sigma(\tau) \equiv \sigma(\alpha \rightarrow \beta)\}) \circ \sigma$



Type Inference Algorithm (cont)

- Case *exp* of
 - $\text{App } (e_1 \ e_2) \rightarrow$
 - Let α be a fresh variable
 - Let $\sigma_1 = \text{infer}(\Gamma, e_1, \alpha \rightarrow \tau)$
 - Let $\sigma_2 = \text{infer}(\sigma(\Gamma), e_2, \sigma(\alpha))$
 - Return $\sigma_2 \circ \sigma_1$



Type Inference Algorithm (cont)

- Case *exp* of
 - If e_1 then e_2 else $e_3 \rightarrow$
 - Let $\sigma_1 = \text{infer}(\Gamma, e_1, \text{bool})$
 - Let $\sigma_2 = \text{infer}(\sigma\Gamma, e_2, \sigma_1(\tau))$
 - Let $\sigma_3 = \text{infer}(\sigma_2 \circ \sigma_1(\Gamma), e_2, \sigma_2 \circ \sigma(\tau))$
 - Return $\sigma_3 \circ \sigma_2 \circ \sigma_1$



Type Inference Algorithm (cont)

- Case *exp* of
 - let $x = e_1$ in $e_2 \rightarrow$
 - Let α be a fresh variable
 - Let $\sigma_1 = \text{infer}(\Gamma, e_1, \alpha)$
 - 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)

- Case *exp* of
 - let rec $x = e_1$ in $e_2 \rightarrow$
 - 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 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”)

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

(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 $i = 1, \dots, n$?



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 $S = \{(s_1 = t_1), (s_2 = t_2), \dots, (s_n = t_n)\}$ be a unification problem.
- Case $S = \{ \}$: $\text{Unif}(S) = \text{Identity function}$ (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 $s = f(q_1, \dots, q_m)$ and $t = f(r_1, \dots, r_m)$ (same f , same $m!$), then $\text{Unif}(S) = \text{Unif}(\{(q_1, r_1), \dots, (q_m, r_m)\} \cup S')$
- **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 $\varphi = \{x \rightarrow t\}$
 - $\text{Unif}(S) = \text{Unif}(\varphi(S')) \circ \{x \rightarrow t\}$
 - Let $\psi = \text{Unif}(\varphi(S'))$
 - $\text{Unif}(S) = \{x \rightarrow \psi(t)\} \circ \psi$
 - Note: $\{x \rightarrow a\} \circ \{y \rightarrow b\} = \{y \rightarrow (\{x \rightarrow a\}(b))\} \circ \{x \rightarrow a\}$ 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 $\{x \rightarrow g(y, y)\}$
 - 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 $\{x \rightarrow g(y, y)\}$

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



Example

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

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



Example

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

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



Example

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

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



Example

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

- Unify $\{(f(g(y, y)) = f(g(f(z), y)))\}$
 - $\{x \rightarrow g(y, y)\} =$
Unify $\{(g(y, y) = g(f(z), y))\}$ ○ $\{x \rightarrow g(y, y)\}$



Example

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

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



Example

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

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



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 $\{(g(y, y) = g(f(z), y))\} \circ \{x \rightarrow g(y, y)\} =$
Unify $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\}$



Example

- x, y, z variables, f, g constructors
- Unify $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\} = ?$



Example

- x, y, z variables, f, g constructors
- $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\}$ is non-empty
- Unify $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\} = ?$



Example

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



Example

- x, y, z variables, f, g constructors
- Pick a pair: $(y = f(z))$
- Eliminate y with $\{y \rightarrow f(z)\}$
- Unify $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\} =$
Unify $\{(f(z) = f(z))\}$
 - $\{y \rightarrow f(z)\} \circ \{x \rightarrow g(y, y)\} =$Unify $\{(f(z) = f(z))\}$
 - $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\}$



Example

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



Example

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

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



Example

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



Example

- x, y, z variables, f, g constructors
- Pick a pair: $(f(z) = f(z))$
- Delete
- Unify $\{(f(z) = f(z))\}$
 - $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\} =$
Unify $\{\}$ ○ $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\}$



Example

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



Example

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



Example

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

$$\begin{aligned} f(x) &= f(g(f(z), y)) \\ \rightarrow f(g(f(z), f(z))) &= f(g(f(z), f(z))) \end{aligned}$$

$$\begin{aligned} g(y, y) &= x \\ \rightarrow g(f(z), f(z)) &= g(f(z), f(z)) \end{aligned}$$



Example of Failure: Decompose

- $\text{Unify}\{(f(x,g(y)) = f(h(y),x))\}$
- Decompose: $(f(x,g(y)) = f(h(y),x))$
- $= \text{Unify}\{(x = h(y)), (g(y) = x)\}$
- Orient: $(g(y) = x)$
- $= \text{Unify}\{(x = h(y)), (x = g(y))\}$
- Eliminate: $(x = h(y))$
- $\text{Unify}\{(h(y) = g(y))\} \circ \{x \rightarrow h(y)\}$
- No rule to apply! Decompose fails!



Example of Failure: Occurs Check

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

Major Phases of a Compiler

Source Program

Lex

Tokens

Parse

Abstract Syntax

Semantic
Analysis

Symbol Table

Translate

Intermediate
Representation

Optimize

Optimized IR

Instruction
Selection

Unoptimized Machine-
Specific Assembly Language

Optimize

Optimized Machine-Specific
Assembly Language

Emit code

Assembly Language

Assembler

Relocatable
Object Code

Linker

Machine
Code



Meta-discourse

- Language Syntax and Semantics
- Syntax
 - Regular Expressions, DFSA's and NDFSA's
 - Grammars
- Semantics
 - Natural Semantics
 - Transition Semantics



Language Syntax

- Syntax is the description of which strings of symbols are meaningful expressions in a language
- It takes more than syntax to understand a language; need meaning (semantics) too
- Syntax is the entry point



Syntax of English Language

- Pattern 1

Subject	Verb
<i>David</i>	<i>sings</i>
<i>The dog</i>	<i>barked</i>
<i>Susan</i>	<i>yawned</i>

- Pattern 2

Subject	Verb	Direct Object
<i>David</i>	<i>sings</i>	<i>ballads</i>
<i>The professor</i>	<i>wants</i>	<i>to retire</i>
<i>The jury</i>	<i>found</i>	<i>the defendant guilty</i>



Elements of Syntax

- Character set – previously always ASCII, now often 64 character sets
- Keywords – usually reserved
- Special constants – cannot be assigned to
- Identifiers – can be assigned to
- Operator symbols
- Delimiters (parenthesis, braces, brackets)
- Blanks (aka white space)



Elements of Syntax

- Expressions

if ... then begin ... ; ... end else begin ... ; ... end

- Type expressions

typexpr₁ -> typexpr₂

- Declarations (in functional languages)

let pattern₁ = expr₁ in expr

- Statements (in imperative languages)

a = b + c

- Subprograms

let pattern₁ = let rec inner = ... in expr



Elements of Syntax

- Modules
- Interfaces
- Classes (for object-oriented languages)



Lexing and Parsing

- Converting strings to abstract syntax trees done in two phases
 - **Lexing:** Converting string (or streams of characters) into lists (or streams) of tokens (the “words” of the language)
 - Specification Technique: Regular Expressions
 - **Parsing:** Convert a list of tokens into an abstract syntax tree
 - Specification Technique: BNF Grammars



Formal Language Descriptions

- Regular expressions, regular grammars, finite state automata
- Context-free grammars, BNF grammars, syntax diagrams
- Whole family more of grammars and automata – covered in automata theory



Grammars

- Grammars are formal descriptions of which strings over a given character set are in a particular language
- Language designers write grammar
- Language implementers use grammar to know what programs to accept
- Language users use grammar to know how to write legitimate programs