Programming Languages and Compilers (CS 421)

William Mansky

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

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

Type Inference - Example

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

$$[f:\varphi \rightarrow \varepsilon;\ x:\beta] \vdash f:\zeta \rightarrow \varphi \qquad [f:\varphi \rightarrow \varepsilon;\ x:\beta] \vdash x:\zeta$$

$$\ldots \qquad [f : \varphi \rightarrow \varepsilon;\ x : \beta] \vdash f\ x : \varphi$$

$$[f : \delta;\ x : \beta] \vdash (f\ (f\ x)) : \varepsilon$$

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

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

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

Type Inference - Example

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

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

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

Background for Unification

- Terms (expressions) made from constructors and variables
- 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
- Build up substitution of terms for variables
- More general than just OCaml!

Simple Implementation Background

- Term is var or function symbol (possibly const):

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

- We need to be able to substitute terms for vars:

```
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 in 1, …, n?

Uses for Unification

- Type inference and type checking
- Pattern matching as in OCAML
- Logic Programming – Prolog and others
- Simple parsing
- Maude (CS422/CS476)

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 cases

Unification Algorithm

- Case $S = \{(s, t)\} \cup S'$: Four cases
- Delete: if $s = t$ (they are the same term) then $\mathrm{Unif}(S) = \mathrm{Unif}(S')$
- 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 is some variable x, and s is not a variable, $\mathrm{Unif}(S) = \mathrm{Unif}(\{(x,s)\} \cup S')$

Unification Algorithm

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

Tricks for Efficient Unification

- Don't return substitution, 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)))
- Eliminate: {x |-> g(y,f(y))}
- S -> {(x, g(y,z)), (x, g(y,f(y)))}
- -> {(g(y,f(y)), g(y,z))}

- With substitution {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 substitution {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 substitution $\{x \mapsto g(y,f(y))\}$

Example

- x,y,z variables, f,g constructors
- Pick a pair: (y, y)
- S -> {(y, y), (f(y), z)}
- With substitution {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 substitution $\{x \mid\!\rightarrow g(y,f(y))\}$

Example

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

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

- With substitution {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 substitution {x $|\rightarrow$ g(y,f(y))}

Example

- x,y,z variables, f,g constructors
- Pick a pair: (z, f(y))
- S -> {(z, f(y))}
- With substitution $\{x \mapsto g(y,f(y))\}$

Example

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

- With substitution

$\{x \mid\!\rightarrow \{z \mid\!\rightarrow f(y)\}\ (g(y,f(y)))\ \} \circ \{z \mid\!\rightarrow f(y)\}$

Example

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

With $\{x \mapsto g(y,f(y))\} \circ \{(z \mapsto f(y))\}$

Example

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

Solved by $\{x \mapsto g(y,f(y))\} \circ \{(z \mapsto f(y))\}$

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

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 $|\rightarrow$ 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.

Most General Unifier

- Unify (f(x,y),f(y,z))
- Two possible solutions:
  - σ_1 = {y |-> x, z |-> x}
  - σ_2 = {x |-> int, y |-> int, z |-> int}
- Which solution is better? The more general one
- σ_2 = {x |-> int} o σ_1, so σ_1 is more general
- Our algorithm produces Most General Unifier