Programming Languages and Compilers (CS 421)

Grigore Rosu
2110 SC, UIUC
http://courses.engr.illinois.edu/cs421

Slides by Elsa Gunter, based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Objective

- Learn type inference and the distinction between type inference and type checking
- Understand how type inference algorithmically reduces to unification
- Learn unification and how to implement it

Type Inference

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:

  [ ] |- (fun x -> fun f -> f (f x)) : α

- Second approximate: use fun rule

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

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

Type Inference - Example

- Third approximate: use fun rule

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

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

Type Inference - Example

- Fourth approximate: use app rule

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

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

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

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

Type Inference - Example

- Current subst:

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

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

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

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

Type Inference - Example

- Current subst:

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

- Solves subproof; return on layer

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

Type Inference - Example

- Current subst:

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

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

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

Type Inference Algorithm

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([$x : \alpha$] + Γ, e, β)
    - Return Unify{$\sigma(\tau) \equiv \sigma(\alpha \rightarrow \beta)$} $\circ$ 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, $\alpha \rightarrow \tau$)
    - 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($\sigma_2 \circ \sigma_1(\Gamma)$,$e_2$,$\sigma_2 \circ \sigma_1(\tau)$)
    - 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([$x : \alpha$] + 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 $\sigma(\alpha)$
- Need an algorithm for Unify

Unification

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; the latter are also called constants
- 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
- …

Unification Algorithm

- Let $S = \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$ be a unification problem.
- Case $S = \{\ \}$: Unify(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 Unify(S) = Unify(S')
- Decompose: if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ (same f, same m!), then Unify(S) = Unify($\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S'$)
- Orient: if t = x is a variable, and s is not a variable, Unify(S) = Unify($\{(x,s)\} \cup S'$)

Unification Algorithm

- Eliminate: if s is var x, and x does not occur in t (occurs check), then
  - Let $\varphi = x \mapsto t$
  - Let ψ = Unify(φ(S'))
  - Unify(S) = $\{x \mapsto \psi(t)\} \circ \psi$
    - Recall how composition of substitutions works. For example (suppose y not in a):
      $\{x \mapsto a\} \circ \{y \mapsto b\} =$
      $\{y \mapsto (\{x \mapsto a\}(b))\} \circ \{x \mapsto a\}$

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(x),x)
- S -> {(x,h(x)), (x,g(x))}
- No rules apply.