

# Programming Languages and Compilers (CS 421)

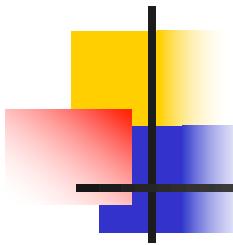


Elsa L Gunter

2112 SC, UIUC

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

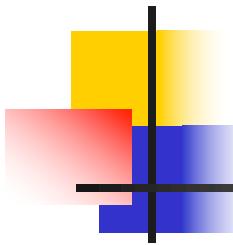
Based in part on slides by Mattox Beckman, as updated  
by Vikram Adve and Gul Agha



# 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

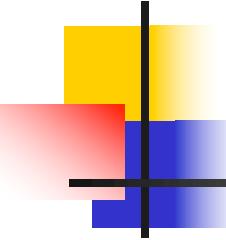
---

## ■ Modus Ponens

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

## • Application

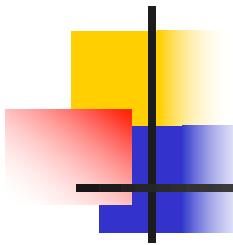
$$\frac{\Gamma \vdash e_1 : \alpha \rightarrow \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash (e_1 \ e_2) : \beta}$$



# Mea Culpa

---

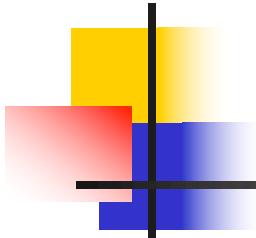
- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only meta-variable in the logic)
- Would need:
  - Object level type variables and some kind of type quantification
  - **let** and **let rec** rules to introduce polymorphism
  - Explicit rule to eliminate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



# Support for Polymorphic Types

---

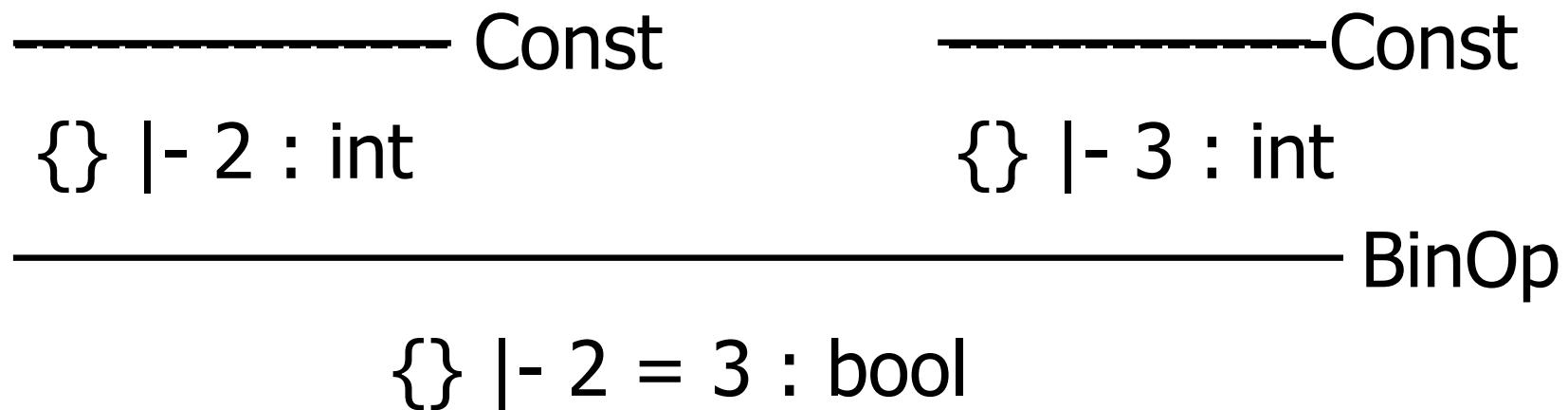
- Monomorphic Types ( $\tau$ ):
  - Basic Types: int, bool, float, string, unit, ...
  - Type Variables:  $\alpha, \beta, \gamma, \delta, \varepsilon$
  - Compound Types:  $\alpha \rightarrow \beta$ , int \* string, bool list, ...
- Polymorphic Types:
  - Monomorphic types  $\tau$
  - Universally quantified monomorphic types
  - $\forall \alpha_1, \dots, \alpha_n . \tau$
  - Can think of  $\tau$  as same as  $\forall . \tau$

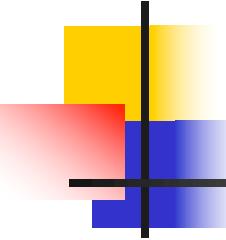


Example: {} |- 2 = 3 : bool

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

# Instance: 'a specialized to int

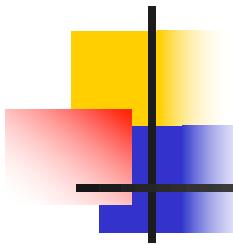




# Polymorphic Example

---

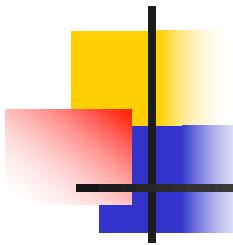
- Assume additional constants and primitive operators:
- $(=) : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \text{bool}$
- $\text{hd} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha$
- $\text{tl} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $\text{is\_empty} : \forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$
- $(::) : \forall \alpha. \alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $[] : \forall \alpha. \alpha \text{ list}$



# Example FreeVars Calculations

---

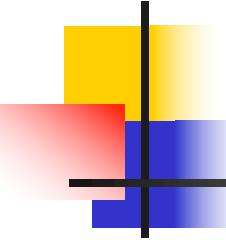
- $\text{Vars}(`a \rightarrow (\text{int} \rightarrow `b) \rightarrow `a) = \{\text{'a}, \text{'b}\}$
- $\text{FreeVars} (\text{All } `b. \text{ 'a} \rightarrow (\text{int} \rightarrow `b) \rightarrow \text{'a}) =$
- $\{\text{'a}, \text{'b}\} - \{\text{'b}\} = \{\text{'a}\}$
- $\text{FreeVars} \{x :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



# Support for Polymorphic Types

---

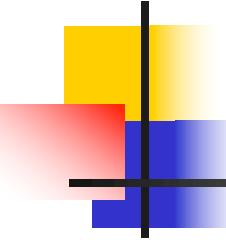
- Typing Environment  $\Gamma$  supplies polymorphic types (which will often just be monomorphic) for variables
- Free variables of monomorphic type just type variables that occur in it
  - Write  $\text{FreeVars}(\tau)$
- Free variables of polymorphic type removes variables that are universally quantified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



# Monomorphic to Polymorphic

---

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



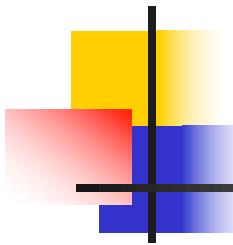
# Polymorphic Typing Rules

---

- A *type judgement* has the form

$$\Gamma \vdash \text{exp} : \tau$$

- $\Gamma$  uses **polymorphic** types
  - $\tau$  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)

---

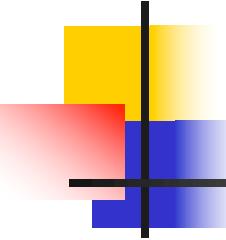
Variable axiom:

$$\frac{}{\Gamma \vdash x : \varphi(\tau)} \quad \text{if } \Gamma(x) = \forall \alpha_1, \dots, \alpha_n . \tau$$

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

$$\frac{}{\Gamma \vdash x : \tau} \quad \text{if } \Gamma(x) = \tau$$

- Constants treated same way



# Polymorphic Let and Let Rec

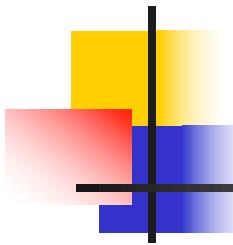
---

- let rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \{x: \text{Gen}(\tau_1, \Gamma)\} + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : \tau_2}$$

- let rec rule:

$$\frac{\{x: \tau_1\} + \Gamma \vdash e_1 : \tau_1 \quad \{x: \text{Gen}(\tau_1, \Gamma)\} + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let rec } x = e_1 \text{ in } e_2) : \tau_2}$$



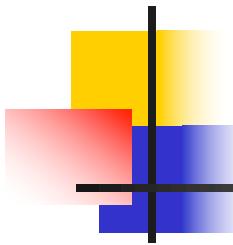
# Fun Rule Stays the Same

---

- fun rule:

$$\frac{\{x: \tau_1\} + \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

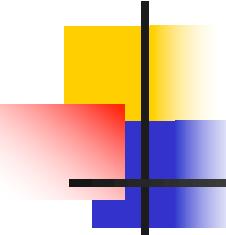
- Types  $\tau_1, \tau_2$  monomorphic
- Function argument must always be used at same type in function body



# Two Problems

---

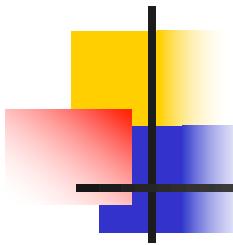
- Type checking
  - Question: Does exp.  $e$  have type  $\tau$  in env  $\Gamma$ ?
  - Answer: Yes / No
  - Method: Type **derivation**
- Typability
  - Question Does exp.  $e$  have **some type** in env.  $\Gamma$ ?  
If so, what is it?
  - Answer: Type  $\tau$  / error
  - Method: Type **inference**



# Type Inference - Outline

---

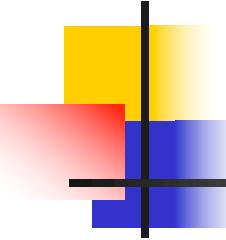
- Begin by assigning a type variable as the type of the whole expression
- Decompose the expression into component expressions
- Use typing rules to generate constraints on components and whole
- Recursively find substitution that solves typing judgment of first subcomponent
- Apply substitution to next 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

---

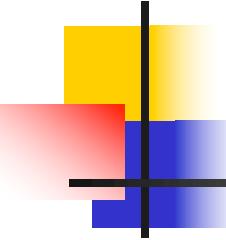
- First approximate:

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

- Second approximate: use fun rule

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

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



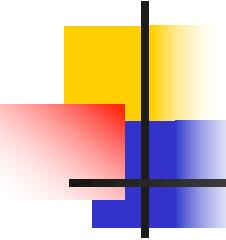
# Type Inference - Example

---

- Third approximate: use fun rule

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

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

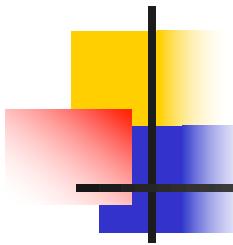


# Type Inference - Example

- Fourth approximate: use app rule

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

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



# Type Inference - Example

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

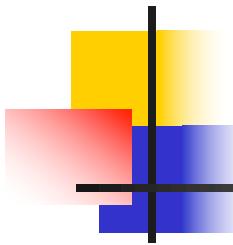
$$\frac{\{f:\delta; x:\beta\} \vdash f : \varphi \rightarrow \varepsilon \quad \{f:\delta; x:\beta\} \vdash fx : \varphi}{}$$

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

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

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

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



# Type Inference - Example

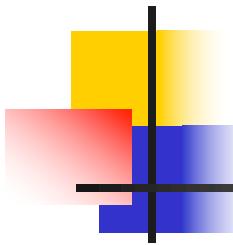
- Current subst:  $\{\delta \equiv \varphi \rightarrow \varepsilon\}$

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

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

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

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

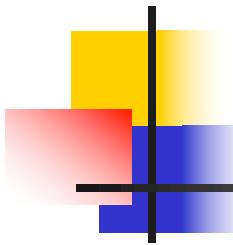


# Type Inference - Example

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

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

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



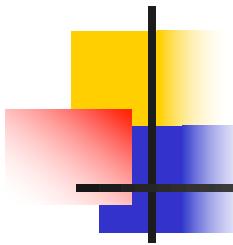
# Type Inference - Example

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

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

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

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



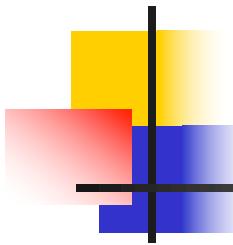
# Type Inference - Example

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

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

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

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

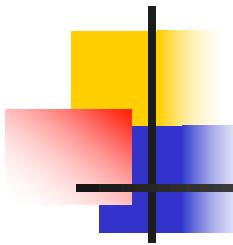


# Type Inference - Example

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

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

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



# Type Inference - Example

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

...

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

...

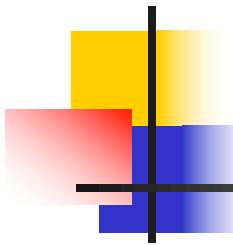
$$\frac{}{\{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi}$$

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

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

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

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



# Type Inference - Example

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

...

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

...

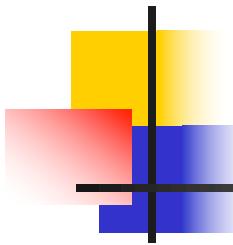
$$\frac{}{\{f:\varphi \rightarrow \varepsilon; x:\beta\} \vdash f x : \varphi}$$

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

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

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

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

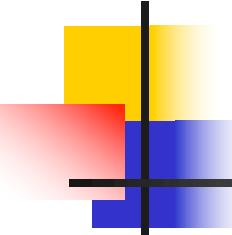


# Type Inference - Example

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

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

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



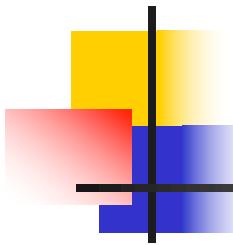
# Type Inference - Example

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

...

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

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



# 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

...

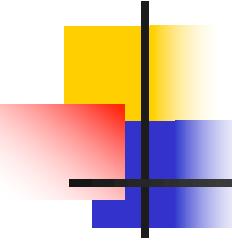
---

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

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

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

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



# Type Inference - Example

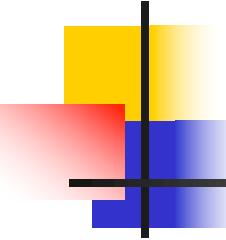
- Current subst:

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

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

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

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



# Type Inference - Example

---

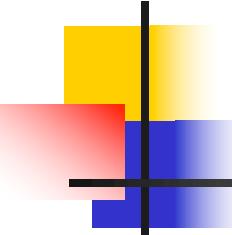
- Current subst:

$$\begin{aligned}\{\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta)), \\ \gamma \equiv ((\beta \rightarrow \beta) \rightarrow \beta), \varepsilon \equiv \beta, \zeta \equiv \beta, \varphi \equiv \beta, \delta \equiv \beta \rightarrow \beta\}\end{aligned}$$

- Solves subproof; return on layer

---

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



# Type Inference - Example

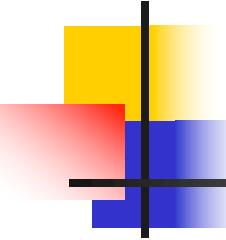
---

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

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

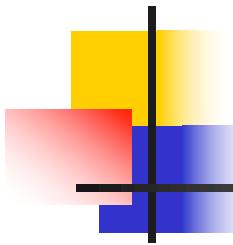


# Type Inference Algorithm

---

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

- $\Gamma$  is a typing environment (giving polymorphic types to expression variables)
- $e$  is an expression
- $\tau$  is a type (with type variables),
- $\sigma$  is a substitution of types for type variables
- Idea:  $\sigma$  is the constraints on type variables necessary for  $\Gamma \vdash e : \tau$
- Should have  $\sigma(\Gamma) \vdash e : \sigma(\tau)$  valid

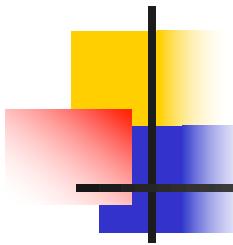


# Type Inference Algorithm

---

`infer ( $\Gamma$ ,  $exp$ ,  $\tau$ ) =`

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



# Example of inference with Var Rule

---

Instance  $\{`a \rightarrow `w\}$  ( $`w$  a fresh variable)

---

$\{x: \text{All } `a. (`a * `b) \text{ list}, y: \text{All. } `b\} \vdash x : (\text{int} * \text{string}) \text{ list}$

$\text{freshInstance}(\text{All } `a. (`a * `b) \text{ list}) = (`w * `b) \text{ list}$

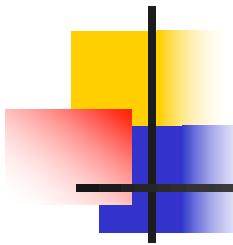
$\text{Unify } \{((\text{int} * \text{string}) \text{ list} = (`w * `b) \text{ list})\} = \{`w \rightarrow \text{int}, `b \rightarrow \text{string}\}$

After substitution:

Instance  $\{`a \rightarrow \text{int}\}$

---

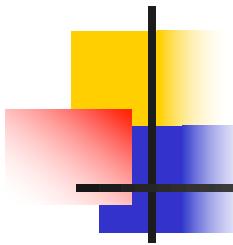
$\{x: \text{All } `a. (`a * \text{string}) \text{ list}, y: \text{All. string}\} \vdash x: (\text{int} * \text{string}) \text{ list}$



# Type Inference Algorithm (cont)

---

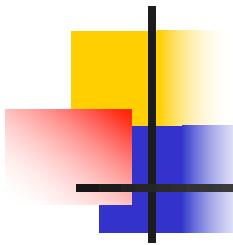
- Case  $\exp$  of
  - App ( $e_1 e_2$ ) -->
    - Let  $\alpha$  be a fresh variable
    - Let  $\sigma_1 = \text{infer}(\Gamma, e_1, \alpha \rightarrow \tau)$
    - Let  $\sigma_2 = \text{infer}(\sigma_1(\Gamma), e_2, \sigma_1(\alpha))$
    - Return  $\sigma_2 \circ \sigma_1$



# Type Inference Algorithm (cont)

---

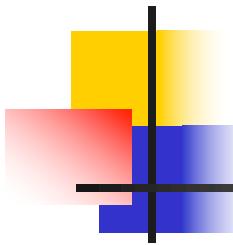
- Case  $\exp$  of
  - If  $e_1$  then  $e_2$  else  $e_3 \rightarrow$ 
    - Let  $\sigma_1 = \text{infer}(\Gamma, e_1, \text{bool})$
    - Let  $\sigma_2 = \text{infer}(\sigma_1\Gamma, e_2, \sigma_1(\tau))$
    - Let  $\sigma_3 = \text{infer}(\sigma_2 \circ \sigma_1(\Gamma), e_3, \sigma_2 \circ \sigma_1(\tau))$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



# Type Inference Algorithm (cont)

---

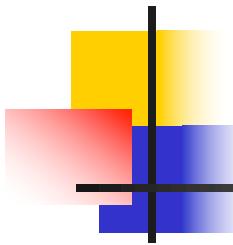
- Case  $\exp$  of
  - $\text{let } x = e_1 \text{ in } e_2 \rightarrow$ 
    - Let  $\alpha$  be a fresh variable
    - Let  $\sigma_1 = \text{infer}(\Gamma, e_1, \alpha)$
    - Let  $\sigma_2 = \text{infer}(\{x:\text{GEN}(\sigma_1(\Gamma), \sigma_1(\alpha))\} + \sigma_1(\Gamma), e_2, \sigma_1(\tau))$
    - Return  $\sigma_2 \circ \sigma_1$



# Type Inference Algorithm (cont)

---

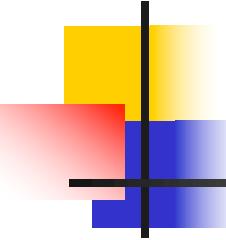
- Case  $\exp$  of
  - $\text{let rec } x = e_1 \text{ in } e_2 \rightarrow$ 
    - Let  $\alpha$  be a fresh variable
    - Let  $\sigma_1 = \text{infer}(\{x: \alpha\} + \Gamma, e_1, \alpha)$
    - Let  $\sigma_2 = \text{infer}(\{x: \text{GEN}(\sigma_1(\Gamma), \sigma_1(\alpha))\} + \sigma_1(\Gamma)\}, e_2, \sigma_1(\tau))$
    - Return  $\sigma_2 \circ \sigma_1$



# Type Inference Algorithm (cont)

---

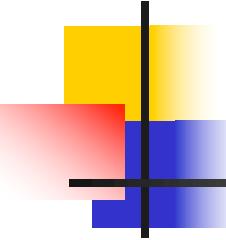
- To infer a type, introduce `type_of`
- Let  $\alpha$  be a fresh variable
- $\text{type\_of}(\Gamma, e) =$ 
  - Let  $\sigma = \text{infer}(\Gamma, e, \alpha)$
  - Return  $\sigma(\alpha)$
- Need an algorithm for `Unif`



# 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

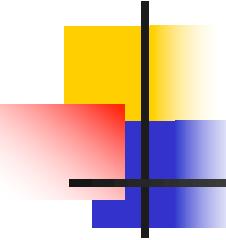


# Simple Implementation Background

---

```
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

---

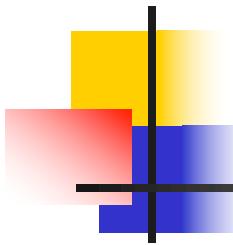
Given a set of pairs of terms (“equations”)

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

(the *unification problem*) does there exist  
a substitution  $\sigma$  (the *unification solution*)  
of terms for variables such that

$$\sigma(s_i) = \sigma(t_i),$$

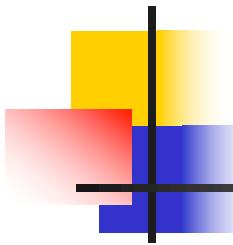
for all  $i = 1, \dots, n$ ?



# Uses for Unification

---

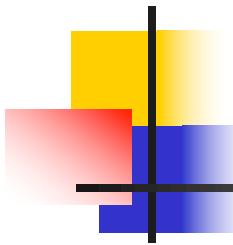
- Type Inference and type checking
- Pattern matching as in OCaml
  - Can use a simplified version of algorithm
- Logic Programming - Prolog
- Simple parsing



# Unification Algorithm

---

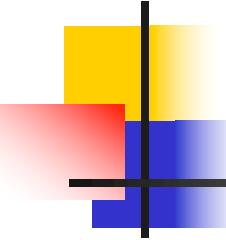
- Let  $S = \{(s_1 = t_1), (s_2 = t_2), \dots, (s_n = t_n)\}$  be a unification problem.
- Case  $S = \{\}$ :  $\text{Unif}(S) = \text{Identity function}$  (i.e., no substitution)
- Case  $S = \{(s, t)\} \cup S'$ : Four main steps



# Unification Algorithm

---

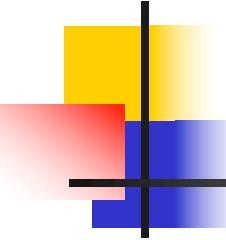
- **Delete:** if  $s = t$  (they are the same term)  
then  $\text{Unif}(S) = \text{Unif}(S')$
- **Decompose:** if  $s = f(q_1, \dots, q_m)$  and  
 $t = f(r_1, \dots, r_m)$  (same  $f$ , same  $m!$ ), then  
 $\text{Unif}(S) = \text{Unif}(\{(q_1, r_1), \dots, (q_m, r_m)\} \cup S')$
- **Orient:** if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



# Unification Algorithm

---

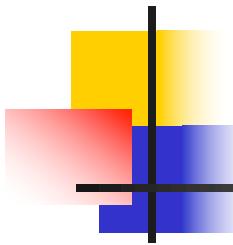
- **Eliminate:** if  $s = x$  is a variable, and  $x$  does not occur in  $t$  (the occurs check), then
  - Let  $\varphi = \{x \rightarrow t\}$ 
    - $\text{Unif}(S) = \text{Unif}(\varphi(S')) \circ \{x \rightarrow t\}$
  - Let  $\psi = \text{Unif}(\varphi(S'))$
  - $\text{Unif}(S) = \{x \rightarrow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

---

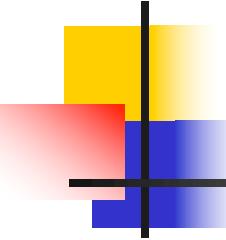
- Don't return substitution, rather do it incrementally
- Make substitution be constant time
  - Requires implementation of terms to use mutable structures (or possibly lazy structures)
  - We won't discuss these



# Example

---

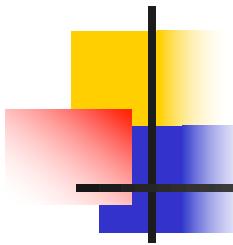
- $x, y, z$  variables,  $f, g$  constructors
- Unify  $\{(f(x) = f(g(f(z), y))), (g(y, y) = x)\} = ?$



# Example

---

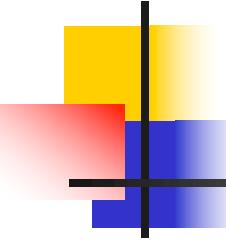
- $x, y, z$  variables,  $f, g$  constructors
- $S = \{(f(x) = f(g(f(z), y))), (g(y, y) = x)\}$  is nonempty
- Unify  $\{(f(x) = f(g(f(z), y))), (g(y, y) = x)\} = ?$



# Example

---

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

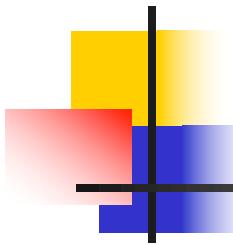


# Example

---

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

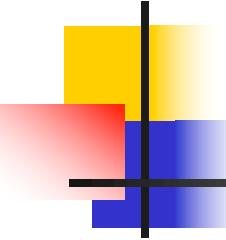
by Orient



# Example

---

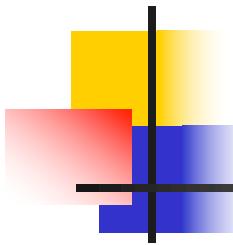
- $x, y, z$  variables,  $f, g$  constructors
- Unify  $\{(f(x) = f(g(f(z), y))), (x = g(y, y))\} = ?$



# Example

---

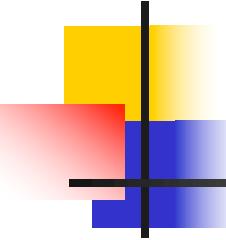
- $x, y, z$  variables,  $f, g$  constructors
- $\{(f(x) = f(g(f(z), y))), (x = g(y, y))\}$  is non-empty
- Unify  $\{(f(x) = f(g(f(z), y))), (x = g(y, y))\} = ?$



# Example

---

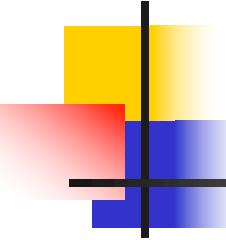
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(x = g(y, y))$
- Unify  $\{(f(x) = f(g(f(z), y))), (x = g(y, y))\} = ?$



# Example

---

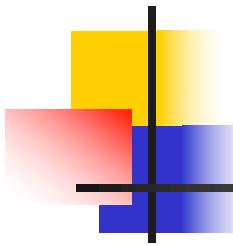
- $x,y,z$  variables,  $f,g$  constructors
- Pick a pair:  $(x = g(y,y))$
- Eliminate  $x$  with substitution  $\{x \rightarrow g(y,y)\}$ 
  - Check:  $x$  not in  $g(y,y)$
- Unify  $\{(f(x) = f(g(f(z),y))), (x = g(y,y))\} = ?$



# Example

---

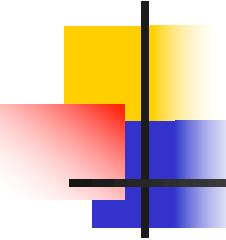
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(x = g(y, y))$
- Eliminate  $x$  with substitution  $\{x \rightarrow g(y, y)\}$
- Unify  $\{(f(x) = f(g(f(z), y))), (x = g(y, y))\} =$   
Unify  $\{(f(g(y, y)) = f(g(f(z), y)))\}$ 
  - o  $\{x \rightarrow g(y, y)\}$



# Example

---

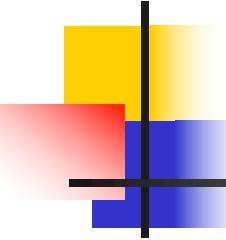
- $x, y, z$  variables,  $f, g$  constructors
- Unify  $\{(f(g(y,y)) = f(g(f(z),y)))\}$ 
  - $\{x \rightarrow g(y,y)\} = ?$



# Example

---

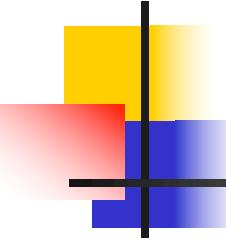
- $x, y, z$  variables,  $f, g$  constructors
- $\{(f(g(y,y)) = f(g(f(z),y)))\}$  is non-empty
- Unify  $\{(f(g(y,y)) = f(g(f(z),y)))\}$ 
  - o  $\{x \rightarrow g(y,y)\} = ?$



# Example

---

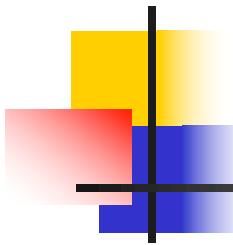
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(f(g(y,y)) = f(g(f(z),y)))$
- Unify  $\{(f(g(y,y)) = f(g(f(z),y)))\}$ 
  - o  $\{x \rightarrow g(y,y)\} = ?$



# Example

---

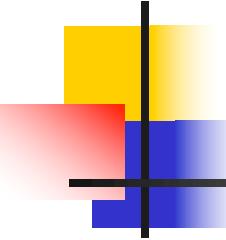
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(f(g(y,y)) = f(g(f(z),y)))$
- Decompose:  $(f(g(y,y)) = f(g(f(z),y)))$   
becomes  $\{(g(y,y) = g(f(z),y))\}$
- Unify  $\{(f(g(y,y)) = f(g(f(z),y)))\}$ 
  - o  $\{x \rightarrow g(y,y)\} =$   
 $\text{Unify } \{(g(y,y) = g(f(z),y))\} \circ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



# Example

---

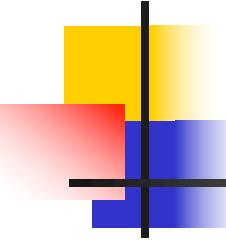
- $x, y, z$  variables,  $f, g$  constructors
- $\{(g(y,y) = g(f(z),y))\}$  is non-empty
- Unify  $\{(g(y,y) = g(f(z),y))\}$ 
  - o  $\{x \rightarrow g(y,y)\} = ?$



# Example

---

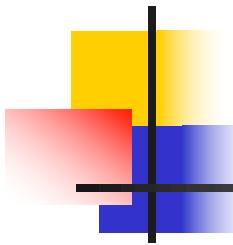
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(g(y,y) = g(f(z),y))$
- Unify  $\{(g(y,y) = g(f(z),y))\}$ 
  - o  $\{x \rightarrow g(y,y)\} = ?$



# Example

---

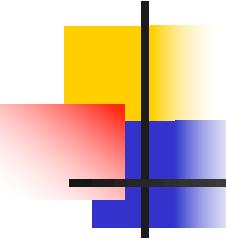
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(f(g(y,y)) = f(g(f(z),y)))$
- Decompose:  $(g(y,y)) = g(f(z),y))$  becomes  
 $\{(y = f(z)); (y = y)\}$
- Unify  $\{(g(y,y) = g(f(z),y))\} \circ \{x \rightarrow g(y,y)\} =$   
Unify  $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y,y)\}$



# Example

---

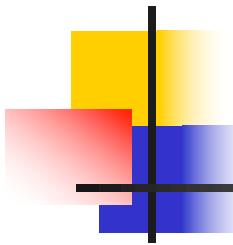
- $x, y, z$  variables,  $f, g$  constructors
- Unify  $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\} = ?$



# Example

---

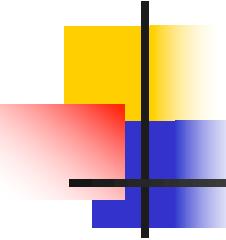
- $x, y, z$  variables,  $f, g$  constructors
- $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\}$  is non-empty
- Unify  $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\} = ?$



# Example

---

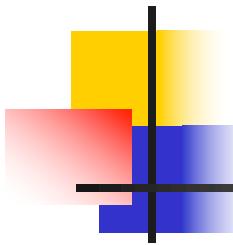
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(y = f(z))$
- Unify  $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\} = ?$



# Example

---

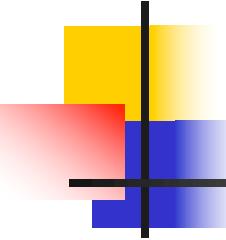
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(y = f(z))$
- Eliminate  $y$  with  $\{y \rightarrow f(z)\}$
- Unify  $\{(y = f(z)); (y = y)\} \circ \{x \rightarrow g(y, y)\} =$   
Unify  $\{(f(z) = f(z))\}$ 
  - o  $(\{y \rightarrow f(z)\} \circ \{x \rightarrow g(y, y)\}) =$   
Unify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



# Example

---

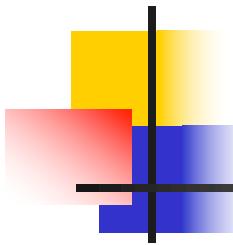
- $x, y, z$  variables,  $f, g$  constructors
- Unify  $\{(f(z) = f(z))\}$ 
  - o  $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\} = ?$



# Example

---

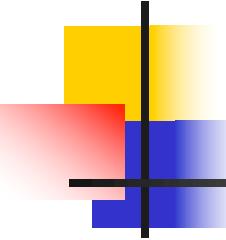
- $x, y, z$  variables,  $f, g$  constructors
- $\{(f(z) = f(z))\}$  is non-empty
- Unify  $\{(f(z) = f(z))\}$ 
  - o  $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\} = ?$



# Example

---

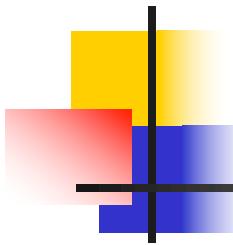
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(f(z) = f(z))$
- Unify  $\{(f(z) = f(z))\}$ 
  - o  $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\} = ?$



# Example

---

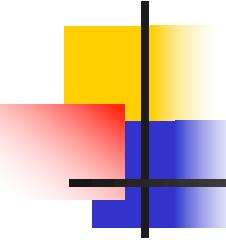
- $x, y, z$  variables,  $f, g$  constructors
- Pick a pair:  $(f(z) = f(z))$
- Delete
- Unify  $\{(f(z) = f(z))\}$ 
  - o  $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\} =$
  - Unify  $\{\}$  o  $\{y \rightarrow f(z); x \rightarrow g(f(z), f(z))\}$



# Example

---

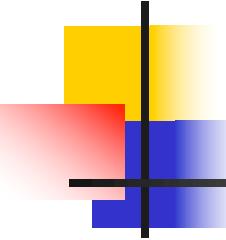
- $x, y, z$  variables,  $f, g$  constructors
- Unify  $\{ \} \circ \{ y \rightarrow f(z); x \rightarrow g(f(z), f(z)) \} = ?$



# Example

---

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



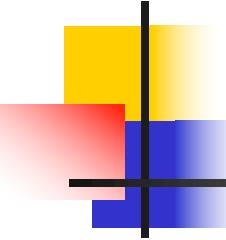
## Example

---

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

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

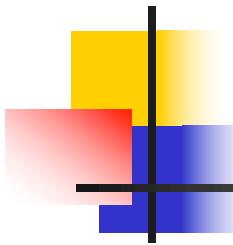
$$\begin{aligned} g(\quad y \quad, \quad y \quad) &= \quad x \\ \rightarrow g(f(z), f(z)) &= g(f(z), f(z)) \end{aligned}$$



# 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  $\{(h(y) = g(y))\} \circ \{x \rightarrow h(y)\}$
- 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.