Programming Languages and Compilers (CS 421)

Sasa Misailovic

4110 SC, UIUC

https://courses.engr.illinois.edu/cs421/fa2024/CS421C

Based on slides by Elsa Gunter, which are based in part on previous slides by Mattox Beckman and updated by Vikram Adve and Gul Agha

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

Types of these expressions?

```
x + 1

x +. 1.0

(fun x -> x + 1)

(fun x -> x + 1)  5

let y = x in y + 1
```

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 composition of all substitutions** to original type variable to get answer

Solving Analogy: Einstein's Riddle

There are 5 houses painted five different colors.
In each house lives a person with a different nationality.
These five owners drink a certain type of beverage, smoke a certain brand of cigar, and keep a certain pet.
No owners have the same pet, smoke the same brand of cigar, or drink the same beverage.

The Brit lives in the red house
The Swede keeps dogs as pets
The Dane drinks tea
The green house is on the left of the white house
… (see the rest on the Internet)

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

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

- First approximate: **Give type to full expr**

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

Type Inference - Example

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

- First approximate: **Give type to full expr**

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

- Second approximate: **use fun rule**

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

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

Type Inference - Example

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

- Third approximate: **use fun rule**

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

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

Type Inference - Example

$$\frac{\Gamma \mid\!- e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \mid\!- e_2 : \tau_1}{\Gamma \mid\!- (e_1\ e_2) : \tau_2}$$

- Fourth approximate: **use app rule**

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

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

Type Inference - Example

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

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

$$\frac{\Gamma \mid\!- e_1 : \tau_1 \to \tau_2 \quad \Gamma \mid\!- e_2 : \tau_1}{\Gamma \mid\!- (e_1\ e_2) : \tau_2}$$

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

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

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

Type Inference - Example

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

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

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

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**

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

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

Type Inference - Example

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

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

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

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

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

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, becomes: $\gamma \equiv ((\beta \to \beta) \to \beta)$

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

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

Type Inference - Example

- Current subst:

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

- Solves subproof; return one layer

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

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

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

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

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

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

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

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

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

Type Inference Algorithm

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

- Γ is a typing environment (giving polymorphic types to expression variables)
- e is an expression
- τ is a type (with type variables),
- σ is a <u>substitution</u> of types for type variables
- **Idea**: σ represents the constraints on type variables necessary for Γ |- e : τ
- Should have $\sigma(\Gamma)$ |- e : $\sigma(\tau)$ valid
  - Slight abuse of notation: $\sigma(\Gamma)$ is substitution σ applied to all terms in the environment $\Gamma = \{x: \tau \ldots\}$ (i.e., $\sigma(\Gamma) = \{x: \sigma(\tau) \ldots\}$).

Type Inference Algorithm (All in one!)

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* : φ by the constant rules
  - fun *x* -> *e* -->
    - Let α, β be fresh variables
    - Let σ = infer ({x: α} + Γ, *e*, β)
    - Return Unify({$\sigma(\tau) \equiv \sigma(\alpha \to \beta)$}) $\circ\ \sigma$
  - App (e_1 e_2) -->
    - Let α be a fresh variable
    - Let σ_1 = infer(Γ, e_1, $\alpha \to \tau$)
    - Let σ_2 = infer($\sigma(\Gamma)$, e_2, $\sigma(\alpha)$)
    - Return $\sigma_2 \circ \sigma_1$
  - If e_1 then e_2 else e_3 -->
    - Let σ_1 = infer(Γ, e_1, bool)
    - Let σ_2 = infer($\sigma\Gamma$, e_2, $\sigma_1(\tau)$)
    - Let σ_3 = infer($\sigma_2 \circ \sigma_1(\Gamma)$,$e_2$,$\sigma_2 \circ \sigma(\tau)$)
    - Return $\sigma_3 \circ \sigma_2 \circ \sigma_1$
  - 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 $\sigma_2 \circ \sigma_1$
  - let rec *x* = e_1 in e_2 -->
    - Let α be a fresh variable
    - Let σ_1 = infer({x: α} + 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

infer (Γ, *exp*, τ) =

- Case *exp* of
  - Var *v* --> return Unify{$\tau \equiv$ freshInstance(Γ(*v*))}
    - Replace all quantified type vars by fresh ones
  - Const *c* --> return Unify{$\tau \equiv$ freshInstance φ } where Γ |- *c* : φ by the constant rules
  - …

Inference Example (Repeat)

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

$$\frac{}{\{f:\delta;\ x:\beta\}\ |\!-\ f : \varphi \to \varepsilon}$$

infer (Γ, *exp*, τ) =

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

Inference Example (Repeat)

- What do do we do here?

$$\frac{}{\{f: \forall\, \delta.\ \delta \rightarrow \delta\ ;\ x{:}\beta\}\ |\text{-}\ f : \varphi \rightarrow \varepsilon}$$

- And here?

$$\{f: \forall\, \varepsilon.\ \varepsilon \rightarrow \varepsilon\ ;\ x{:}\beta\}\ |\text{-}\ f : \varphi \rightarrow \varepsilon$$

infer (Γ, *exp*, τ) =

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

Type Inference Algorithm

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

infer (Γ, *exp*, τ) =

- Case *exp* of
  - …
  - fun *x* -> e -->
    - Let α, β be fresh variables
    - Let σ = infer ($\{x: \alpha\} + \Gamma$, e, β)
    - Return Unify($\{\sigma(\tau) \equiv \sigma(\alpha \to \beta)\}$) $\circ\ \sigma$

Inference Example (Repeat)

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

- Third approximate: **use fun rule**

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

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

infer (Γ, *exp*, τ) =

- Case *exp* of
  - fun *x* -> *e* -->
    - Let β, γ be fresh variables
    - Let σ = infer ($\{x: \beta\} + \Gamma$, *e*, γ)
    - Return Unify($\{\sigma(\tau) \equiv \sigma(\beta \rightarrow \gamma)\}$) $\circ\ \sigma$

Type Inference Algorithm (cont)

$$\frac{\Gamma \mid\!- \; e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \mid\!- \; e_2 : \tau_1}{\Gamma \mid\!- \; (e_1 \; e_2) : \tau_2}$$

- Case *exp* of
  - …
  - App $(e_1 \; e_2)$ -->
    - Let α 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 - Example

$$\frac{\Gamma \mid\!- e_1 : \tau_1 \to \tau_2 \quad \Gamma \mid\!- e_2 : \tau_1}{\Gamma \mid\!- (e_1\ e_2) : \tau_2}$$

- Fourth approximate: **use app rule**

$$\frac{\{f{:}\delta;\ x{:}\beta\} \mid\!- f : \varphi \to \varepsilon \quad \{f{:}\delta;\ x{:}\beta\} \mid\!- f\ x : \varphi}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}$$

- Case *exp* of
  - App (e_1 e_2) -->
    - Let α be a fresh variable
    - Let σ_1 = infer(Γ, e_1, $\alpha \to \tau$)
    - Let σ_2 = 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 -->
    - Let σ_1 = infer(Γ, e_1, bool)
    - Let σ_2 = infer($\sigma_1(\Gamma)$, e_2, $\sigma_1(\tau)$)
    - Let σ_3 = infer($\sigma_2 \circ \sigma_1(\Gamma)$,$e_2$,$\sigma_2 \circ \sigma(\tau)$)
    - Return $\sigma_3 \circ \sigma_2 \circ \sigma_1$

- If_then_else rule:

$$\frac{\Gamma \mid\!- e_1 : \text{bool} \quad \Gamma \mid\!- e_2 : \tau \quad \Gamma \mid\!- e_3 : \tau}{\Gamma \mid\!- (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}$$

Type Inference Algorithm (cont)

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

- let rule:

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

Type Inference Algorithm (cont)

- Case *exp* of
  - let rec x = e_1 in e_2 -->
    - Let α be a fresh variable
    - Let σ_1 = infer($\{x: \alpha\} + \Gamma$, e_1, α)
    - Let σ_2 = infer($\{x:\text{GEN}(\sigma_1(\alpha), \sigma_1(\Gamma))\}$ $+ \sigma_1(\Gamma)$, e_2, $\sigma_1(\tau)$)
    - Return $\sigma_2 \circ \sigma_1$

- let rec rule:

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

Type Inference Algorithm (cont)

- To infer a type, introduce type_of
- Let α be a fresh variable
- type_of (Γ, *e*) =

```
    let α be a fresh variable in
    let σ = infer (Γ, e, α)
    in σ (α)
```

- Need substitution!
- Need an algorithm for Unif!

Background for Unification
Reminder: Type Terms

- Terms 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
- Substitution of terms for variables (function subst)
  - E.g. replace “x” with “y” in [“z”, “x”]

Substitution Simple Implementation

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

(* substitution of variable name with “residue” in the “term” *)
let rec subst var_name residue term =
    match term with
        Variable name ->
            if var_name = name
            then residue
            else term
       | Constructor (c, tys) ->
            let newt = List.map (subst var_name residue) tys
            in Constructor (c, newt);;
```

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

$$\sigma(s_i) \text{ is the same as } \sigma(t_i),$$

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

- ***Think of these pairs as $\{(``s_1 = t_1"), (``s_2 = t_2"), \ldots, (``s_n = t_n")\}$**
  - This is the notation we're going to use in the example

Uses for Unification

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

Unification Algorithm

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

Unification Algorithm for $S = \{(s = t)\} \cup S'$

- Delete: if s is t (s and t are the same term) then
  $\mathrm{Unif}(S) = \mathrm{Unif}(S')$

- Decompose: if s is $f(q_1, \ldots, q_m)$ and t is $f(r_1, \ldots, r_m)$ (**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 x (a variable), and s is not a variable,
  $\mathrm{Unif}(S) = \mathrm{Unif}(\{(x = s)\} \cup S')$

Unification Algorithm for $S = \{(s = t)\} \cup S'$

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

  - Be careful when composing substitutions:
  - $\{x \to a\} \circ \{y \to b\} =$

    $\{y \to (\{x \to a\}(b))\} \circ \{x \to a\}$ if y not in a
  - So we can also write the last step as

    $\mathrm{Unif}(S) = \psi \circ \{x \to t\}$

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

- Unify {(f(x) = f(g(f(z),y))), (g(y,y) = x)} = ?
- For example:
  - X list = (Z list * Y) list
  - (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)

Unification Algorithm for $S = \{(s = t)\} \cup S'$

- Delete: if s is t (s and t are the same term) then
  Unif(S) = Unif(S')
- Decompose: if s is f $(q_1, \ldots, q_m)$ and t is f $(r_1, \ldots, r_m)$ (**same f, same m!**), then
  $\text{Unif}(S) = \text{Unif}(\{(q_1 = r_1), \ldots, (q_m = r_m)\} \cup S')$
- Orient: if t is x (a variable), and s is not a variable,
  $\text{Unif}(S) = \text{Unif}(\{(x = s)\} \cup S')$
- Eliminate: if s is x (a variable), and x does not occur in t* then
  - Let $\varphi = \{x \to t\}$
  - Let $\psi = \text{Unif}(\varphi(S'))$
  - $\text{Unif}(S) = \{x \to \psi(t)\} \circ \psi$

- 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))

Unification Algorithm for $S = \{(s = t)\} \cup S'$

- Delete: if s is t (s and t are the same term) then
  Unif(S) = Unif(S')
- Decompose: if s is $f(q_1, \ldots, q_m)$ and t is $f(r_1, \ldots, r_m)$ (**same f, same m!**), then
  $\text{Unif}(S) = \text{Unif}(\{(q_1 = r_1), \ldots, (q_m = r_m)\} \cup S')$
- Orient: if t is x (a variable), and s is not a variable,
  $\text{Unif}(S) = \text{Unif}(\{(x = s)\} \cup S')$
- Eliminate: if s is x (a variable), and x does not occur in t* then
  - Let $\varphi = \{x \to t\}$
  - Let $\psi = \text{Unif}(\varphi(S'))$
  - $\text{Unif}(S) = \{x \to \psi(t)\} \circ \psi$

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

Example

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

Example

- x,y,z variables, f,g constructors
- Pick a pair: (x = g(y,y))
- Eliminate x with substitution {x→ g(y,y)}

- Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} =
  Unify {(f(g(y,y)) = f(g(f(z),y)))}
  o {x→ g(y,y)}

Example

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

- Unify {(f(g(y,y)) = f(g(f(z),y)))}

  $\circ$ {x→ g(y,y)} = ?

Example

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

- Unify {(f(g(y,y)) = f(g(f(z),y)))}

  o {x→ g(y,y)} = ?

Example

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

Unification Algorithm for $S = \{(s = t)\} \cup S'$

- Delete: if s is t (s and t are the same term) then
  Unif(S) = Unif(S')
- Decompose: if s is f $(q_1, \ldots, q_m)$ and t is f $(r_1, \ldots, r_m)$ (**same f, same m!**), then
  Unif(S) = Unif($\{(q_1 = r_1), \ldots, (q_m = r_m)\} \cup S'$)
- Orient: if t is x (a variable), and s is not a variable,
  Unif(S) = Unif ($\{(x = s)\} \cup S'$)
- Eliminate: if s is x (a variable), and x does not occur in t* then
  - Let $\varphi = \{x \to t\}$
  - Let ψ = Unif(φ (S'))
  - Unif(S) = $\{x \to \psi(t)\} \circ \psi$

- Unify {(f(g(y,y)) = f(g(f(z),y)))}

  o {x→ g(y,y)} = ?

Example

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

- Unify {(f(g(y,y)) = f(g(f(z),y)))}

  o {x→ g(y,y)} =

 Unify {(g(y,y) = g(f(z),y))} o {x→ g(y,y)}

Example

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

- Unify {(g(y,y) = g(f(z),y))}

  o {x→ g(y,y)} = ?

Example

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

Unification Algorithm for S = {(s = t)} ∪ S'

- Delete: if s is t (s and t are the same term) then
  Unif(S) = Unif(S')
- Decompose: if s is f (q_1, … , q_m) and t is f (r_1, … , r_m) (**same f, same m!**), then
  Unif(S) = Unif({(q_1= r_1), …, (q_m=r_m)} ∪ S')
- Orient: if t is x (a variable), and s is not a variable,
  Unif(S) = Unif ({(x = s)} ∪ S')
- Eliminate: if s is x (a variable), and x does not occur in t* then
  - Let φ = {x → t}
  - Let ψ = Unif(φ (S'))
  - Unif(S) = {x → ψ(t)} o ψ

- Unify {(g(y,y) = g(f(z),y))}

  o {x→ g(y,y)} = ?

Example

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

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

Example

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

- Unify {(y = f(z)); (y = y)} o {x→ g(y,y)} = ?

Example

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

Example

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

Unification Algorithm for $S = \{(s = t)\} \cup S'$

- Delete: if s is t (s and t are the same term) then
  $Unif(S) = Unif(S')$
- Decompose: if s is $f(q_1, \ldots, q_m)$ and t is $f(r_1, \ldots, r_m)$ (**same f, same m!**), then
  $Unif(S) = Unif(\{(q_1 = r_1), \ldots, (q_m = r_m)\} \cup S')$
- Orient: if t is x (a variable), and s is not a variable,
  $Unif(S) = Unif(\{(x = s)\} \cup S')$
- Eliminate: if s is x (a variable), and x does not occur in t* then
  - Let $\varphi = \{x \to t\}$
  - Let $\psi = Unif(\varphi(S'))$
  - $Unif(S) = \{x \to \psi(t)\} \circ \psi$

Example

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

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

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

Example

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

- Unify {(f(z) = f(z))}
  - o {y → f(z); x→ g(f(z), f(z))} = ?

Example

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

- Unify {(f(z) = f(z))}
  - o {y → f(z); x→ g(f(z), f(z))} = ?

Example

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

- Unify {(f(z) = f(z))}
  o {y → f(z); x→ g(f(z), f(z))} = ?

Unification Algorithm for S = {(s = t)} ∪ S'

- Delete: if s is t (s and t are the same term) then
  Unif(S) = Unif(S')
- Decompose: if s is f (q_1, … , q_m) and t is f (r_1, … , r_m) (**same f, same m!**), then
  Unif(S) = Unif({(q_1= r_1), …, (q_m=r_m)} ∪ S')
- Orient: if t is x (a variable), and s is not a variable,
  Unif(S) = Unif ({(x = s)} ∪ S')
- Eliminate: if s is x (a variable), and x does not occur in t* then
  - Let φ = {x → t}
  - Let ψ = Unif(φ (S'))
  - Unif(S) = {x → ψ(t)} o ψ

Example

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

  o {y → f(z); x→ g(f(z), f(z))} =

  Unify {} o {y → f(z); x→ g(f(z), f(z))}

Example

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

- Unify {} o {y → f(z); x→ g(f(z), f(z))} = ?

Example

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

  {y → f(z); x→ g(f(z), f(z))}

Example

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

  $\{y \to f(z); x \to g(f(z), f(z))\}$

$$f(\quad x \quad) = f(g(f(z), \quad y \quad))$$

$$\to f(g(f(z), f(z))) = f(g(f(z), f(z)))$$

$$g(\; y \;, \; y \;) = \quad x$$

$$\to g(f(z),f(z)) = g(f(z), f(z))$$

Example

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

  {y → f(z); x→ g(f(z), f(z))}

  y → int list, x→ (int list * int list)

  f(x) = f(g(f(z), y))

  → f(g(f(z), f(z))) = f(g(f(z), f(z)))

  (int list * int list) list = (int list * int list) list

  g(y , y) = x

  → g(f(z),f(z)) = g(f(z), f(z))

  (int list * int list) = (int list * int list)

Example of Failure: Decompose

- Unify{(f(x,g(y)) = f(h(y),x))}

  Decompose: (f(x,g(y)) = f(h(y),x))

  = Unify {(x = h(y)), (g(y) = x)}

  Orient: (g(y) = x)

  = Unify {(x = h(y)), (x = g(y))}

  Eliminate: (x = h(y))

  = Unify {(h(y) = g(y))} o {x → 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)}

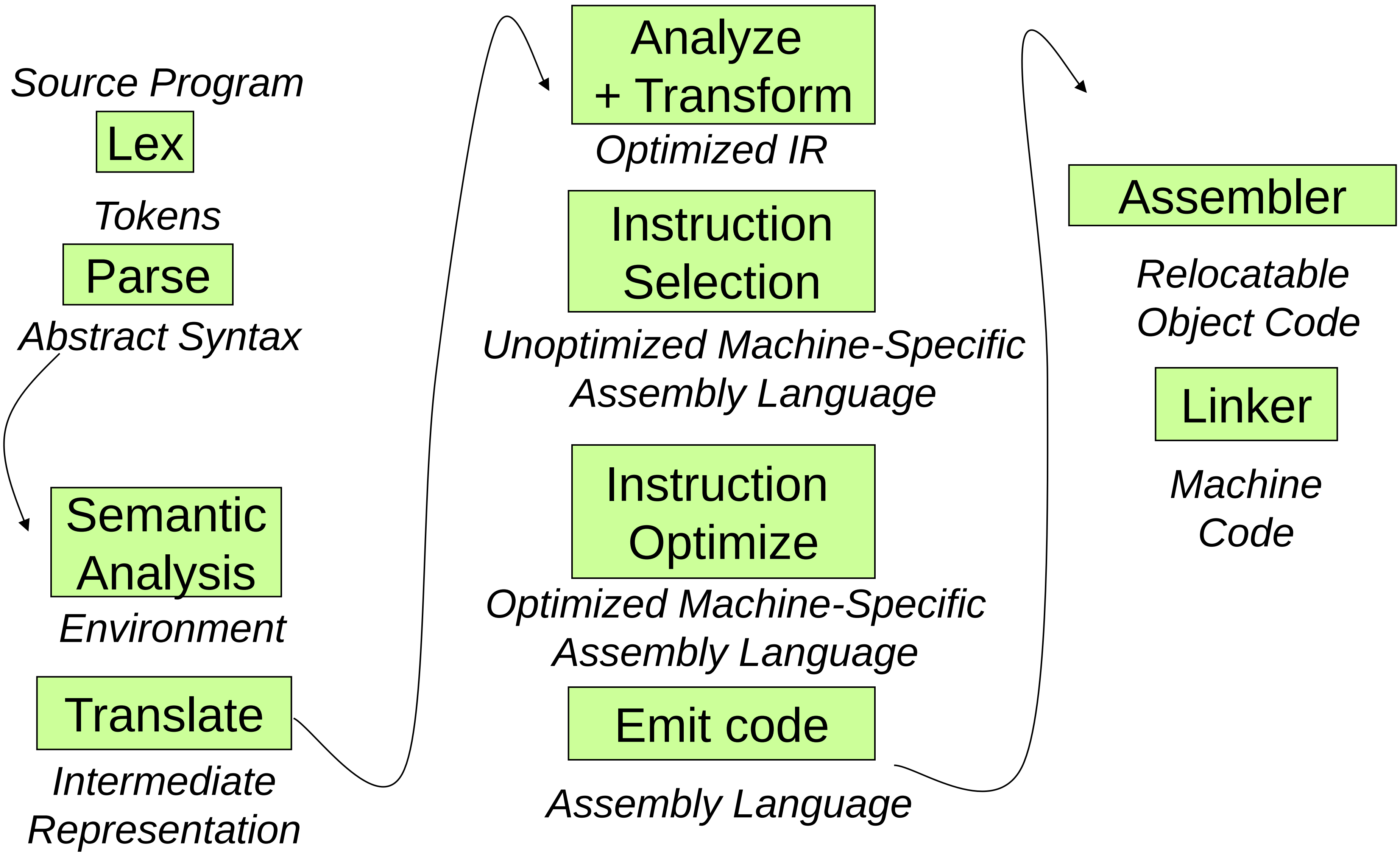
  Orient: (g(y) = x)

- = Unify {(x = h(x)), (x = g(x))}
  - **No rules apply.**

Course Objectives

- New programming paradigm
  - Functional programming
  - Environments and Closures
  - Patterns of Recursion
  - Continuation Passing Style
- Phases of an interpreter / compiler
  - Lexing and parsing
  - Type systems
  - Interpretation
- Programming Language Semantics
  - Lambda Calculus
  - Operational Semantics
  - Axiomatic Semantics

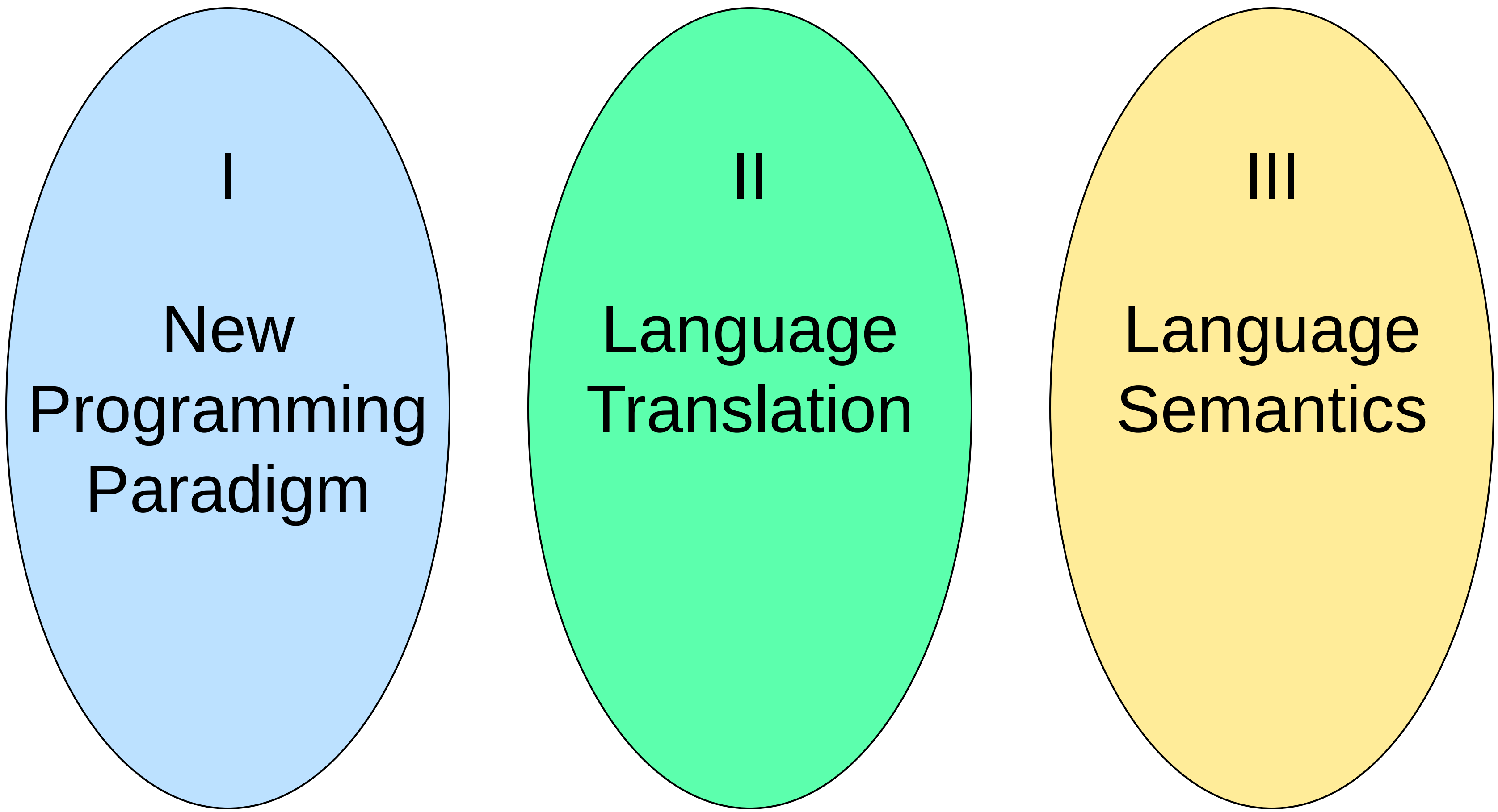
Major Phases of a Compiler



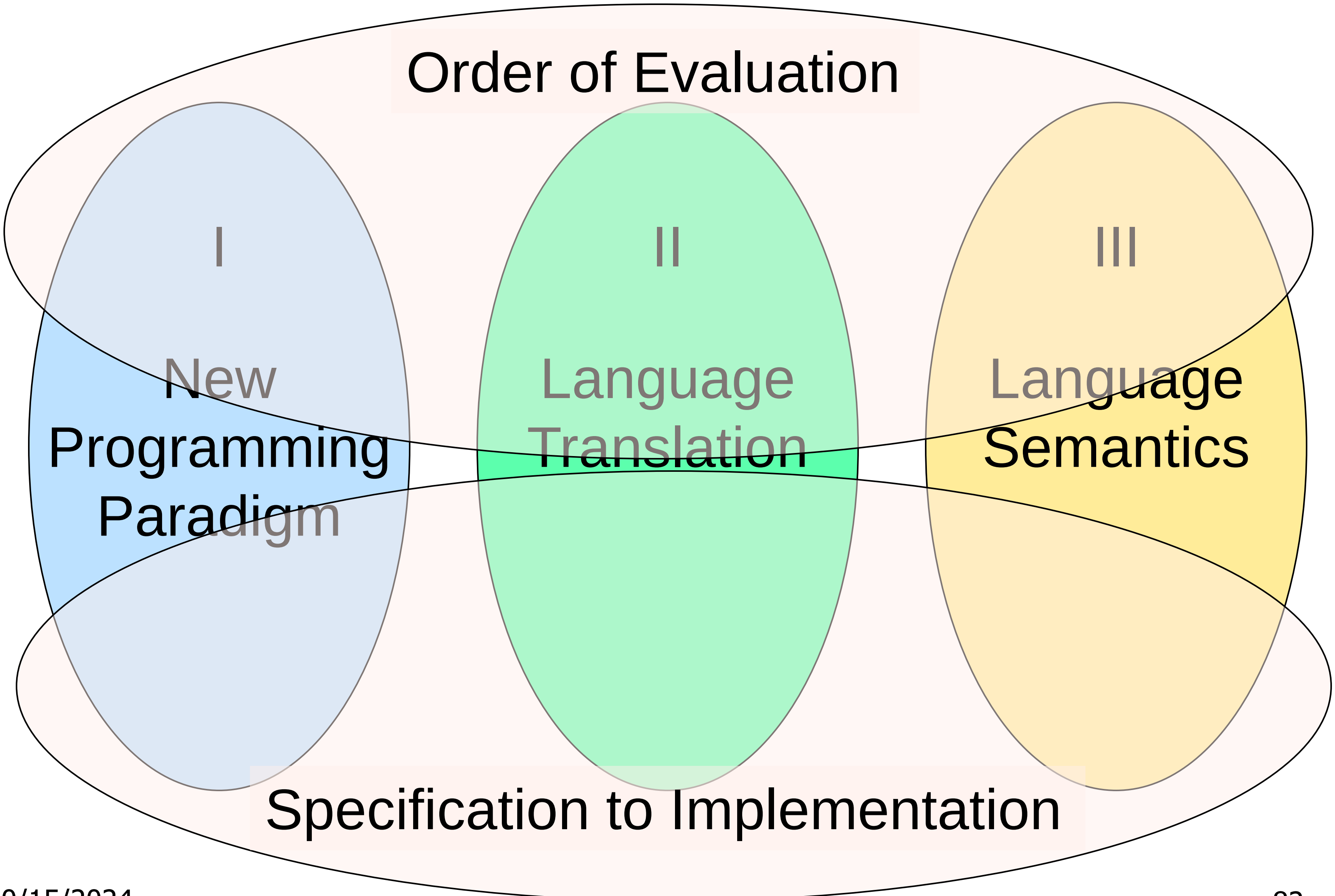
Modified from "Modern Compiler Implementation in ML", by Andrew Appel

Programming Languages & Compilers

Three Main Topics of the Course



Programming Languages & Compilers



Programming Languages & Compilers

I : New Programming Paradigm

- Functional Programming
- Environments and Closures
- Patterns of Recursion
- Continuation Passing Style

Programming Languages & Compilers

II : Language Translation

Lexing and Parsing

Type Systems

Interpretation

Programming Languages & Compilers

III : Language Semantics

Operational Semantics

Lambda Calculus

Axiomatic Semantics

Meta-discourse

- Language Syntax and Semantics
- Syntax
  - Regular Expressions, DFSAs and NDFSAs
  - 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**
David	*sings*
The dog	*barked*
Susan	*yawned*

- Pattern 2

Subject	**Verb**	**Direct Object**
David	*sings*	*ballads*
The professor	*wants*	*to retire*
The jury	*found*	*the defendant guilty*

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

  $typexpr_1$ -> $typexpr_2$

- Declarations (in functional languages)

  let $pattern_1$ = $expr_1$ in *expr*

- Statements (in imperative languages)

  a = b + c

- Subprograms

  let $pattern_1$ = 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