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

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:

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

- 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

- Third approximate: use fun rule

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

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

Type Inference - Example

- Fourth approximate: use app rule

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

Type Inference - Example

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

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

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

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

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

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

Type Inference - Example

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

… $\{f:\varepsilon\rightarrow\varepsilon;\ x:\beta\}$ |- $x:\varepsilon$

… $\{f:\varphi\rightarrow\varepsilon;\ x:\beta\}$ |- $f\ x : \varphi$

$\{f : \delta\ ;\ x : \beta\}$ |- (f (f x)) : ε

$\{x : \beta\}$ |- (fun f -> f (f x)) : γ

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

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

Type Inference - Example

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

… $\{f:\varepsilon\rightarrow\varepsilon;\ x:\beta\}$ |- $x:\varepsilon$

… $\{f:\varphi\rightarrow\varepsilon;\ x:\beta\}$ |- $f\ x : \varphi$

$\{f : \delta\ ;\ x : \beta\}$ |- (f (f x)) : ε

$\{x : \beta\}$ |- (fun f -> f (f x)) : γ

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

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

Type Inference - Example

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

… $\{f:\varepsilon\rightarrow\varepsilon;\ x:\beta\}$ |- $x:\varepsilon$

… $\{f:\varphi\rightarrow\varepsilon;\ x:\beta\}$ |- $f\ x : \varphi$

$\{f : \delta\ ;\ x : \beta\}$ |- (f (f x)) : ε

$\{x : \beta\}$ |- (fun f -> f (f x)) : γ

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

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

Type Inference - Example

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

… $\{f:\varphi\rightarrow\varepsilon;\ x:\beta\}$ |- $f\ x : \varphi$

$\{f : \delta\ ;\ x : \beta\}$ |- (f (f x)) : ε

$\{x : \beta\}$ |- (fun f -> f (f x)) : γ

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

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

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

$\{f : \delta\ ;\ x : \beta\}$ |- (f (f x)) : ε

$\{x : \beta\}$ |- (fun f -> f (f x)) : γ

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

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

Type Inference - Example

- Current subst:

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

- Solves subproof; return one layer

$\{f : \delta\ ;\ x : \beta\}$ |- (f (f x)) : ε

$\{x : \beta\}$ |- (fun f -> f (f x)) : γ

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

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

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

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

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

Type Inference - Example

- Current subst:

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

- Solves subproof; return on layer

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

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

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

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

has_type (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 $\varphi\}$ where Γ |- $c : \varphi$ by the constant rules
  - fun *x* -> *e* -->
    - Let α, β be fresh variables
    - Let σ = infer ($\{x: \alpha\} + \Gamma$, *e*, β)
    - Return Unify($\{\sigma(\tau) \equiv \sigma(\alpha \rightarrow \beta)\}$) $\circ\ \sigma$

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($\sigma(\Gamma)$, e_2, $\sigma(\alpha)$)
    - Return σ_2 o σ_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($\sigma\Gamma$, e_2, $\sigma_1(\tau)$)
    - Let σ_3 = infer($\sigma_2 \circ \sigma_1(\Gamma)$,$e_2$,$\sigma_2 \circ \sigma(\tau)$)
    - Return $\sigma_3 \circ \sigma_2 \circ \sigma_1$

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: 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 (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 σ (α)
- 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 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
- Simple parsing

Unification Algorithm

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

Unification Algorithm

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

Unification Algorithm

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

- 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

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

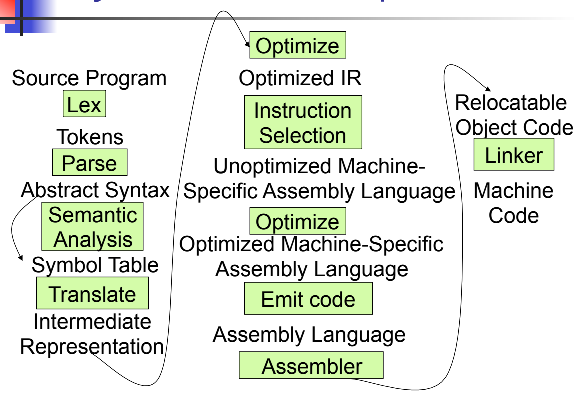
Example of Failure: Decompose

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

Major Phases of a Compiler



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

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