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 $\forall \alpha_1, \ldots, \alpha_n . \tau$
  - Can think of $\tau$ as same as $\forall . \tau$

**Example:** $\{\} |- 2 = 3 : \text{bool}$

$(=) : \text{All } 'a. 'a -> 'a -> \text{bool}$

**Instance:** $'a -> \text{int}$

\[
\begin{array}{c}
\{\} |- (=) : \quad \text{Const} \\
\text{int} -> \text{int} -> \text{bool} \quad \{\} |- 2: \text{int} \\
\{\} |- (=) \ 2 : \text{int} -> \text{bool} \quad \{\} |- 3 : \text{int} \\
\{\} |- 2 = 3 : \text{bool}
\end{array}
\]

**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
  - $\text{FreeVars}(\forall \alpha_1, \ldots, \alpha_n . \tau) = \text{FreeVars}(\tau) - \{\alpha_1, \ldots, \alpha_n\}$
- $\text{FreeVars}(\Gamma) = \text{all FreeVars of types in range of } \Gamma$

**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 $\Gamma$

\[
\text{Gen}(\tau, \Gamma) = \forall \alpha_1, \ldots, \alpha_n . \tau \text{ where } \{\alpha_1, \ldots, \alpha_n\} = \text{freeVars}(\tau) - \text{freeVars}(\Gamma)
\]
Polymorphic Typing Rules

- A type judgement has the form $\Gamma |- \text{exp} : \tau$
- $\Gamma$ 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
  - Worth noting functions again

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

- First approximate:
  $\{ \} |- (\text{fun } x \to \text{fun } f \to f(fx)) : \alpha$

- Second approximate: use fun rule
  $\{ x : \beta \} |- (\text{fun } f \to f(fx)) : \gamma$
  $\{ \} |- (\text{fun } x \to \text{fun } f \to f(fx)) : \alpha$
  $\alpha = (\beta \to \gamma)$

- Third approximate: use fun rule
  $\{ f : \delta ; x : \beta \} |- f(fx) : \epsilon$
  $\{ x : \beta \} |- (\text{fun } f \to f(fx)) : \gamma$
  $\{ \} |- (\text{fun } x \to \text{fun } f \to f(fx)) : \alpha$
  $\alpha = (\beta \to \gamma); \gamma = (\delta \to \epsilon)$
Type Inference - Example

Fourth approximate: use app rule
\[
\{ f : \delta ; x : \beta \} \vdash f : \phi \rightarrow \epsilon \\
\{ f : \delta ; x : \beta \} \vdash f \, x : \phi \\
\{ x : \beta \} \vdash (\text{fun } f \rightarrow f \, (f \, x)) : \gamma \\
\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f \, (f \, x)) : \alpha \\
\alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \epsilon)
\]

Type Inference - Example

Current subst: \( \{ \delta \equiv \phi \rightarrow \epsilon \} \)

... \[
\{ f : \phi \rightarrow \epsilon ; x : \beta \} \vdash f \, x : \phi \\
\{ f : \delta ; x : \beta \} \vdash (f \, (f \, x)) : \epsilon \\
\{ x : \beta \} \vdash (\text{fun } f \rightarrow f \, (f \, x)) : \gamma \\
\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f \, (f \, x)) : \alpha \\
\alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \epsilon)
\]

Type Inference - Example

Current subst: \( \{ \delta \equiv \phi \rightarrow \epsilon \} \)

Var rule: Solve \( \zeta \rightarrow \phi \equiv \phi \rightarrow \epsilon \) Unification
\[
\{ f : \phi \rightarrow \epsilon ; x : \beta \} \vdash f : \zeta \rightarrow \phi \\
\{ f : \phi \rightarrow \epsilon ; x : \beta \} \vdash x : \zeta \\
\{ f : \phi \rightarrow \epsilon ; x : \beta \} \vdash f \, x : \phi \\
\{ f : \delta ; x : \beta \} \vdash (f \, (f \, x)) : \epsilon \\
\{ x : \beta \} \vdash (\text{fun } f \rightarrow f \, (f \, x)) : \gamma \\
\{ \} \vdash (\text{fun } x \rightarrow \text{fun } f \rightarrow f \, (f \, x)) : \alpha \\
\alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \epsilon)
\]
Type Inference - Example

- Current subst: \( \{ \zeta = \varepsilon, \varphi = \varepsilon, \delta = \varepsilon \rightarrow \varepsilon \} \)
- Apply to next sub-proof

\[
\ldots \quad \{ f : \varepsilon \rightarrow \varepsilon; \ x : \beta \} \vdash \ x : \varepsilon
\]
\[
\ldots \quad \{ f : \varphi \rightarrow \varepsilon; \ x : \beta \} \vdash f : \varphi
\]
\[
\{ f : \delta; \ x : \beta \} \vdash (f \ (f \ x)) : \varepsilon
\]
\[
\{ x : \beta \} \vdash (\text{fun} \ f \rightarrow f \ (f \ x)) : \gamma
\]
\[
\{ \} \vdash (\text{fun} \ x \rightarrow \text{fun} \ f \rightarrow f \ (f \ x)) : \alpha
\]
- \( \alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \varepsilon) \)

11/2/19

Type Inference - Example

- Current subst: \( \{ \zeta = \varepsilon, \varphi = \varepsilon, \delta = \varepsilon \rightarrow \varepsilon \} \)
- Var rule: \( \varepsilon = \beta \)

\[
\ldots \quad \{ f : \varepsilon \rightarrow \varepsilon; \ x : \beta \} \vdash x : \varepsilon
\]
\[
\ldots \quad \{ f : \varphi \rightarrow \varepsilon; \ x : \beta \} \vdash f \ x : \varphi
\]
\[
\{ f : \delta; \ x : \beta \} \vdash (f \ (f \ x)) : \varepsilon
\]
\[
\{ x : \beta \} \vdash (\text{fun} \ f \rightarrow f \ (f \ x)) : \gamma
\]
\[
\{ \} \vdash (\text{fun} \ x \rightarrow \text{fun} \ f \rightarrow f \ (f \ x)) : \alpha
\]
- \( \alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \varepsilon) \)

11/2/19

Type Inference - Example

- Current subst: \( \{ \varepsilon = \beta, \ zeta = \beta, \ varphi = \beta, \ delta = \beta \rightarrow \beta \} \)
- Solves subproof; return one layer

\[
\ldots \quad \{ f : \varepsilon \rightarrow \varepsilon; \ x : \beta \} \vdash x : \varepsilon
\]
\[
\ldots \quad \{ f : \varphi \rightarrow \varepsilon; \ x : \beta \} \vdash f \ x : \varphi
\]
\[
\{ f : \delta; \ x : \beta \} \vdash (f \ (f \ x)) : \varepsilon
\]
\[
\{ x : \beta \} \vdash (\text{fun} \ f \rightarrow f \ (f \ x)) : \gamma
\]
\[
\{ \} \vdash (\text{fun} \ x \rightarrow \text{fun} \ f \rightarrow f \ (f \ x)) : \alpha
\]
- \( \alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \varepsilon) \)

11/2/19

Type Inference - Example

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

\[
\{ f : \delta; \ x : \beta \} \vdash (f \ (f \ x)) : \varepsilon
\]
\[
\{ x : \beta \} \vdash (\text{fun} \ f \rightarrow f \ (f \ x)) : \gamma
\]
\[
\{ \} \vdash (\text{fun} \ x \rightarrow \text{fun} \ f \rightarrow f \ (f \ x)) : \alpha
\]
- \( \alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \varepsilon) \)

11/2/19

Type Inference - Example

- Current subst: \( \{ \gamma = ((\beta \rightarrow \beta) \rightarrow \beta), \varepsilon = \beta, \ zeta = \beta, \ varphi = \beta, \ delta = \beta \rightarrow \beta \} \)
- Solves subproof; return one layer

\[
\ldots \quad \{ f : \delta; \ x : \beta \} \vdash (f \ (f \ x)) : \varepsilon
\]
\[
\{ x : \beta \} \vdash (\text{fun} \ f \rightarrow f \ (f \ x)) : \gamma
\]
\[
\{ \} \vdash (\text{fun} \ x \rightarrow \text{fun} \ f \rightarrow f \ (f \ x)) : \alpha
\]
- \( \alpha = (\beta \rightarrow \gamma); \gamma = (\delta \rightarrow \varepsilon) \)

11/2/19
Type Inference - Example

Current subst:
{γ ≡ ((β→β) → β), ε≡β, ζ≡β, ϕ≡β, δ≡β→β}  

Need to satisfy constraint α = (β → γ)  
given subst: α = (β → ((β→β) → β))  

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

Type Inference Algorithm

Let infer (Γ, e, τ) = σ

Γ is a typing environment (giving polymorphic types to expression variables)

e is an expression

τ is a type (with type variables),

α is a substitution of types for type variables

Idea: α is the constraints on type variables necessary for Γ |- e : τ

Should have α(Γ) |- e : α(τ) valid

Type Inference Algorithm (cont)

Case exp of

App (e1 e2) -->

Let α be a fresh variable

Let α1 = infer(Γ, e1, α → τ)

Let α2 = infer(α(Γ), e2, α(α))

Return α2 o α1
Type Inference Algorithm (cont)

Case \( \text{exp} \) of

- If \( e_1 \) then \( e_2 \) else \( e_3 \) -->
  - Let \( \alpha_1 = \text{infer}(\Gamma, e_1, \text{bool}) \)
  - Let \( \alpha_2 = \text{infer}(\alpha_1, \Gamma, e_2, \alpha_1(\tau)) \)
  - Let \( \alpha_3 = \text{infer}(\alpha_2 \circ \alpha_1(\Gamma), e_2, \alpha_2 \circ \alpha_1(\tau)) \)
  - Return \( \alpha_3 \circ \alpha_2 \circ \alpha_1 \)

Type Inference Algorithm (cont)

Case \( \text{exp} \) of

- Let \( x = e_1 \) in \( e_2 \) -->
  - 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 \( \text{exp} \) of

- To infer a type, introduce \( \text{type}_\text{of} \)
  - Let \( \alpha \) be a fresh variable
  - \( \text{type}_\text{of}(\Gamma, e) = \) 
    - 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);;

11/2/19
Unification Problem

Given a set of pairs of terms (“equations”) 
\{(s_1, t_1), (s_2, t_2), ..., (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 \(i = 1, ..., n\)?

Unification Algorithm

- Let \(S = \{(s_1 = t_1), (s_2 = t_2), ..., (s_n = t_n)\}\) 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
  
  **Delete:** if \(s = t\) (they are the same term) then
  \(\text{Unif}(S) = \text{Unif}(S')\)
  
  **Decompose:** if \(s = f(q_1, ..., q_m)\) and
  \(t = f(r_1, ..., r_m)\) (same \(f\), same \(m\!\)), then
  \(\text{Unif}(S) = \text{Unif}(\{(q_1, r_1), ..., (q_m, r_m)\} \cup S')\)
  
  **Orient:** if \(t = x\) is a variable, and \(s\) is not a
  variable, \(\text{Unif}(S) = \text{Unif}(\{(x = s)\} \cup S')\)
  
  **Eliminate:** if \(s = x\) is a variable, and
  \(x\) does not occur in \(t\) (the occurs check), then
  - Let \(\psi = \{x \rightarrow t\}\)
    - \(\text{Unif}(S) = \text{Unif}(\psi(S')) o \{x \rightarrow t\}\)
    - Let \(\psi = \text{Unif}(\psi(S'))\)
    - \(\text{Unif}(S) = \{x \rightarrow \psi(t)\} o \psi\)
  
  Note: \(\{x \rightarrow a\} o \{y \rightarrow b\} = \{y \rightarrow (((x \rightarrow a)(b))) o \{x \rightarrow a\}\) 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

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

Example

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

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

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 \{\((f(z) = f(z))\) \(\circ\) \(y \rightarrow f(z); x \rightarrow g(f(z), f(z))\)\} = ?

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

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

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

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

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

Example of Failure: Decompose
- Unify $\{(f(x, g(y)) = f(h(y), x))\}$
- Decompose: $(f(x, g(y)) = f(h(y), x))$
  - $= \text{Unify } \{(x = h(y)), (g(y) = x)\}$
  - Orient: $(g(y) = x)$
  - $= \text{Unify } \{(x = h(y)), (x = g(y))\}$
  - 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: $(f(x, g(x)) = f(h(x), x))$
  - $= \text{Unify } \{(x = h(x)), (g(x) = x)\}$
  - Orient: $(g(y) = x)$
  - $= \text{Unify } \{(x = h(x)), (x = g(x))\}$
  - 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
- Pattern 2

<table>
<thead>
<tr>
<th>Subject</th>
<th>Verb</th>
<th>Direct Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>sings</td>
<td>ballads</td>
</tr>
<tr>
<td>The dog</td>
<td>barked</td>
<td></td>
</tr>
<tr>
<td>Susan</td>
<td>yawned</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Verb</th>
<th>Direct Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>sings</td>
<td>the defendant</td>
</tr>
<tr>
<td>The professor</td>
<td>wants</td>
<td>to retire</td>
</tr>
<tr>
<td>The jury</td>
<td>found</td>
<td>the defendant</td>
</tr>
</tbody>
</table>

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)
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