Name Analysis

Kristoffer H. Rose    Eva Rose
{krisrose,evarose}@cs.nyu.edu

Compiler Construction (CSCI-GA.2130-001) Spring 2014
NYU Courant Institute

March 3, 2014
Outline

1. Introduction
2. Symbol Tables = Environments
3. HACS
4. Extending hw4 Question 2.2 with Declarations
Introduction

Symbol Tables = Environments

HACS

Extending hw4 Question 2.2 with Declarations

Context

source program

Tokens

Lexical Analysis

Syntax Analysis

Symbol Table

Tree

Semantic Analysis

Intermediate Representation Generator

IR

Optimizer

IR

Code Generator

IR

Machine-Dependent Code Optimizer

target machine code
Example Code

1. `int initial = 32;`
2. `float rate = .8;`
3. `float position = initial + rate * 8;`
Example Abstract Syntax Tree (AST)

```
int = ⟨id, 2⟩ ⟨num, 32⟩

float = ⟨id, 3⟩ ⟨num, .8⟩

float = ⟨id, 1⟩ + ⟨id, 2⟩ * ⟨id, 3⟩ ⟨num, 60⟩
```
Example Abstract Syntax Tree (AST) + “def-use”
Binding Construct

```
int =

Local → Value

Scope
```

Introduction
Symbol Tables = Environments
HACS
Extending hw4 Question 2.2 with Declarations

K.H. Rose, E. Rose
CSCI-GA.2130-001/Spring 2014/Name Analysis (March 3, 2014)
Binding Construct II
**Compile time vs Runtime Values**

![Diagram showing the relationship between compile-time environment, variable use, and run-time stack/heap.](image)

- **Variable Use**
  - Compile-time Environment
- **Variable Definition**
  - Run-time Stack/Heap
- **Memory Location**
  - Stack/Heap
Shadowing

```c
int x = 32;
int y;
{
    float x = .8;
    float y = x + x * 8;
}
y = y + x;
```
Symbol Tables

- Traditional method for managing binders in system.
- Logically one symbol table per scope.
  ... really messy to manage with imperative SDTs!
- We shall fix this!
Symbol Tables

- Traditional method for managing binders in system.
- Logically **one symbol table per scope**.
  ...really messy to manage with imperative SDTs!
- *We shall fix this!*
Symbol Tables

- Traditional method for managing binders in system.
- Logically *one symbol table per scope.*
  ...really messy to manage with imperative SDTs!
- *We shall fix this!*
Binding Construct with Local Symbol Table = Environment

\[
\begin{align*}
\text{int} &= x \\
V &\leadsto \cdots, \\
S &\leadsto e = \{x \mapsto \ldots, \ldots\}
\end{align*}
\]
Binding Construct with Local Symbol Table = Environment II

```latex
\begin{align*}
\text{int} &= V_1 \quad \text{float} = e = \{ x \mapsto \ldots, \ldots \} \\
\text{int} &= V_1 \quad \text{float} = e = \{ y \mapsto \ldots, x \mapsto \ldots, \ldots \} \\
\end{align*}
```
Binding Construct with Local Symbol Table = Environment III

\[
\begin{align*}
\text{int} & = \{x \mapsto \ldots, \ldots\} \\
\text{float} & = \{x \mapsto \ldots, \ldots\} \\
S & = \{x \mapsto \ldots, \ldots\}
\end{align*}
\]
Binding Construct à la HACS

\[
\begin{align*}
\text{int} &= V_1 \\
\text{float} &= V_2 \\
\end{align*}
\]

\[
e = \{ x \mapsto \ldots, \ldots \}
\]

\[
e = \{ y \mapsto \ldots, x \mapsto \ldots, \ldots \}
\]
Binding Construct à la HACS

```
int = V₁

float = float = {x ↦ → ..., ...}

V₂

S

e = {x ↦ → ..., ...}
```
HACS is *Higher-order* Attribute Contraction Schemes

- Traditional:

  \[ P \rightarrow S^* \]
  \[ S \rightarrow \text{int } V = E; \ | \ \text{print } V; \]

- Combine Scoping and Grammar:

  \[ P \rightarrow S \]
  \[ S \rightarrow \text{int } V = E; S \ | \ \text{print } V; S \ | \ \epsilon \]
HACS is *Higher-order* Attribute Contraction Schemes

- **Traditional:**
  
  \[ P \rightarrow S^* \]
  \[ S \rightarrow \text{int } V = E; \mid \text{print } V; \]

- **Combine Scoping and Grammar:**
  
  \[ P \rightarrow S \]
  \[ S \rightarrow \text{int } V = E; S \mid \text{print } V; S \mid \epsilon \]
Outline

1. Introduction
2. Symbol Tables = Environments
3. HACS
4. Extending hw4 Question 2.2 with Declarations
HACS is *Higher-order* Attribute Contraction Schemes II

\[ P \rightarrow S \]
\[ S \rightarrow \text{int } V_x. = E; S^x \mid \text{print } V; S \mid \epsilon \]

sort \( V \mid \text{symbol } [\langle \text{ID} \rangle] \);

sort \( P \mid [\langle S \rangle] \);

sort \( S \mid [\text{int } \langle x:V \rangle = \langle E \rangle; \langle S[x:V] \rangle] \]
\[ \mid [\text{print } \langle V \rangle; \langle S \rangle] \]
\[ \mid [] \];
HACS is *Higher-order* Attribute Contraction Schemes II

\[ P \rightarrow S \]
\[ S \rightarrow \text{int } V_x. = E; S^x | \text{print } V; S | \epsilon \]

**sort** \( V \)  |  **symbol** \([\langle ID \rangle] ;\)

**sort** \( P \)  |  \([\langle S \rangle] ;\)

**sort** \( S \)  |  \([ \text{int } \langle x:V \rangle = \langle E \rangle; \langle S[x:V] \rangle \] \)
|  \([ \text{print } \langle V \rangle; \langle S \rangle ] \)
|  \([ ] ;\)
### Example

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{id} ::= E$</td>
<td>$E.e = S.e; S.sym = \text{id}.sym; S.t = E.t$</td>
</tr>
<tr>
<td>$\mid { S^* }$</td>
<td>$S^*.e = S.e; S.sym = \epsilon; S.t = \epsilon$</td>
</tr>
<tr>
<td>$S^* \rightarrow S_1 S_2^*$</td>
<td>$S_1.e = S^*.e$</td>
</tr>
<tr>
<td>$\mid \epsilon$</td>
<td>$S_2^<em>.e = \text{if } S_1.sym \neq \epsilon \text{ then extend}(S^</em>.e, S_1.sym, S_1.t) \text{ else } \epsilon$</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Symbol Tables = Environments
3. HACS
4. Extending hw4 Question 2.2 with Declarations
... On To The Blackboard!
The following two slides have a sanitized version of the file we wrote together in class.
Extended hw4 Question 2.2 with Declarations (I)

```plaintext
module "edu.nyu.cims.cc.Lecture0303" {
  space [
    \t\n    \r\n  ] ;

  token ID | [a-z]+ (\'_\' [0-9]+)∗ ;
  sort V   | symbol [⟨ID⟩] ;

  sort YesNo | Yes | No ;
  attribute ↑z(YesNo);
  attribute ↓e{V:YesNo};

  sort E    | [ [ ( let ⟨[x:V]⟩⟨E⟩⟨E[x:V]⟩ ) ] ] | [ ⟨V⟩ ]
             | ↑z | scheme Z(E)↓e | scheme Z2(E)↓e ;
```

K.H.Rose, E.Rose
Extending hw4 Question 2.2 with Declarations (II)

\[
Z(\begin{array}{c}
\langle \text{let } x \langle E \#2 \rangle \langle E \#3[x] \rangle \rangle \\
\end{array}) \\
\rightarrow Z2(\begin{array}{c}
\langle \text{let } x \langle E Z(\#2) \rangle \langle E \#3[x] \rangle \rangle \\
\end{array}) ;
\]

\[
Z2(\begin{array}{c}
\langle \text{let } x \langle E \#2 \uparrow z(\#z2) \rangle \langle E \#3[x] \rangle \rangle \\
\end{array}) \\
\rightarrow \begin{array}{c}
\langle \text{let } x \langle E \#2 \rangle \langle E Z(E \#3[x]) \downarrow e\{x: \#z2\} \rangle \rangle \\
\end{array} ;
\]

\[
\begin{array}{c}
\langle \text{let } x \langle E \#2 \rangle \langle E \#3[x] \uparrow z(\#z3) \rangle \rangle \uparrow z(\#z3) ;
\end{array}
\]

\[
Z(\begin{array}{c}
\langle x \rangle \\
\end{array}) \downarrow e\{x: \#z\} \rightarrow \begin{array}{c}
\langle x \rangle \uparrow z(\#z) ;
\end{array}
\]

Note: Notation still subject to change.
Questions?

krisrose@cs.nyu.edu