5. Name Analysis

Eva Rose    Kristoffer Rose

NYU Courant Institute
Compiler Construction (CSCI-GA.2130-001)
http://cs.nyu.edu/courses/fall14/CSCI-GA.2130-001/lecture-5.pdf

October 2, 2014
1. Introduction
2. Programming Language Basics
3. Symbol Tables = Environments
4. HACS
5. Project Milestone 1
Context

source program

Tokens

Lexical Analysis

Syntax Analysis

Symbol Table

Semantic Analysis

Tree

Intermediate Representation Generator

Tree

Optimizer

IR

Code Generator

IR

Machine-Dependent Code Optimizer

target machine code
Example Code

```c
int initial = 32;
float rate = .8;
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⟩
```

<table>
<thead>
<tr>
<th>id</th>
<th>lexeme</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>position</td>
</tr>
<tr>
<td>2</td>
<td>initial</td>
</tr>
<tr>
<td>3</td>
<td>rate</td>
</tr>
</tbody>
</table>

Eva Rose, Kristoffer Rose
Example Abstract Syntax Tree (AST) + “def-use”
Introduction

Programming Language Basics

Symbol Tables = Environments

HACS

Project Milestone 1
What’s in a Name?

**Identifier** name that identifies an entity

**Variable** abstract notion referring to particular storage location
What’s in a Name?

**Identifier**  name that identifies an entity

**Variable**  abstract notion referring to particular *storage location*
Introduction Programming Language Basics Symbol Tables = Environments HACS Project Milestone 1

What Is Known?

Declaration gives type (etc.) of name.

Definition gives value (etc.) of name.
What Is Known?

**Declaration** gives **type** *(etc.)* of name.

**Definition** gives **value** *(etc.)* of name.
**What to Call?**

**Procedure**  any callable entity.

**Function**  callable entity that “returns” a value.

**Method**  callable entity tied to class or object.
What to Call?

Procedure  any callable entity.
Function   callable entity that “returns” a value.
Method     callable entity tied to class or object.
**What to Call?**

- **Procedure** any callable entity.
- **Function** callable entity that “returns” a value.
- **Method** callable entity tied to class or object.
What is Passed?

Actual parameters the values that occur in a call.

Formal parameters the variables used to refer to the parameters inside procedure.
What is Passed?

Actual parameters the values that occur in a call.

Formal parameters the variables used to refer to the parameters inside procedure.
How is it Passed?

Call-by-Value  actual parameter values computed before call.

Call-by-Reference  actual parameter must be variable which is aliased with formal parameter.

Call-by-Name  actual parameter text executed in context of formal parameter.

Call-by-Need  like Call-by-Value but evaluation delayed until first use.

Lazy  just evaluate sufficiently to create observed parts of data structures.
How is it Passed?

**Call-by-Value** actual parameter values computed before call.

**Call-by-Reference** actual parameter must be variable which is aliased with formal parameter.

**Call-by-Name** actual parameter text executed in context of formal parameter.

**Call-by-Need** like Call-by-Value but evaluation delayed until first use.

**Lazy** just evaluate sufficiently to create observed parts of data structures.
How is it Passed?

Call-by-Value  actual parameter values computed before call.

Call-by-Reference  actual parameter must be variable which is aliased with formal parameter.

Call-by-Name  actual parameter text executed in context of formal parameter.

Call-by-Need  like Call-by-Value but evaluation delayed until first use.

Lazy  just evaluate sufficiently to create observed parts of data structures.
How is it Passed?

**Call-by-Value** actual parameter values computed *before* call.

**Call-by-Reference** actual parameter must be variable which is *aliased* with formal parameter.

**Call-by-Name** actual parameter *text* executed in context of formal parameter.

**Call-by-Need** like Call-by-Value but evaluation *delayed* until first use.

Lazy: just evaluate sufficiently to create observed parts of data structures.
How is it Passed?

Call-by-Value actual parameter values computed before call.
Call-by-Reference actual parameter must be variable which is aliased with formal parameter.
Call-by-Name actual parameter text executed in context of formal parameter.
Call-by-Need like Call-by-Value but evaluation delayed until first use.
Lazy just evaluate sufficiently to create observed parts of data structures.
Static vs Dynamic

**Declarations**  Address is fixed vs runtime allocated.

**Classes**  Shared vs per instance.

**Scopes**  Tied to program “blocks” (lexical) vs runtime stack.
Static vs Dynamic

**Declarations**  Address is fixed vs runtime allocated.

**Classes**  Shared vs per instance.

**Scopes**  Tied to program “blocks” (lexical) vs runtime stack.
Static vs Dynamic

**Declarations**  Address is fixed vs runtime allocated.

**Classes**  Shared vs per instance.

**Scopes**  Tied to program “blocks” (lexical) vs runtime stack.
Compile time vs Runtime Values

Variable Use ➔ Variable Definition ➔ Memory Location

- Compile-time Environment
- Run-time Stack/Heap
Environment vs State

- Environment:
  - names
  - locations

- State:
  - names
  - locations
  - values
Environment vs State

- **Environment**
  - names
  - locations
  - values

- **State**
Environment vs State

environment

names locations

state values

static (compile-time)
Environment vs State

- **Environment**
  - Names
  - Locations
  - Values
  - Dynamic (Run-time)

- **State**
  - Names
  - Locations
  - Values
  - Dynamic (Run-time)
Environment vs State

- Environment
  - names
  - locations
  - compiler

- State
  - values
Environment vs State

- **Environment**
  - Names
  - Values
  - Locations

- **State**
  - Run-time
Focus of Class... 

We are concerned with... 

- **Static** scoping. 
- Environments. 
- Planning runtime state.
Focus of Class.

We are concerned with...

- **Static scoping**.
- **Environments**.
- **Planning runtime state**.
Focus of Class... 

We are concerned with... 

- **Static** scoping. 
- **Environments**. 
- **Planning runtime state**. 
Static (Lexical) Scoping

int = Local Value Scope
Formal Parameters

function

Param

Scope
Exercise 1.6.1

```c
int w, x, y, z;
int i = 4; int j = 5;
{
    int j = 7; i = 6; w = i + j;
}
x = i + j;
{
    int i = 8; y = i + j;
}
z = i + j;
```
#define a (x+1)
int x = 2;

void b() {
    x = a; printf("%d\n", x);
}
void c() {
    int x = 1; printf("%d\n", a);
}
void main() { b(); c(); }
1. Introduction

2. Programming Language Basics

3. Symbol Tables = Environments

4. HACS

5. Project Milestone 1
Symbol Tables

- Traditional method for managing binders in system.
- Logically one symbol table per scope.
  - Really a tree: common ancestor blocks can be shared.
  - Can get messy if hash-table is used.
- Interferes with semantic rules/actions.
- *We shall fix this!* It is a little different from the book.
### Symbol Tables

- Traditional method for managing binders in system.
- Logically one symbol table per scope.
  - Really a tree: common ancestor blocks can be shared.
  - Can get messy if hash-table is used.
- Interferes with semantic rules/actions.
- *We shall fix this! It is a little different from the book.*
Symbol Tables

- Traditional method for managing binders in system.
- Logically one symbol table per scope.
  - Really a tree: common ancestor blocks can be shared.
  - Can get messy if hash-table is used.
- Interferes with semantic rules/actions.
- We shall fix this! It is a little different from the book.
Symbol Tables

- Traditional method for managing binders in system.
- Logically one symbol table per scope.
  - Really a tree: common ancestor blocks can be shared.
  - Can get messy if hash-table is used.
- Interferes with semantic rules/actions.
- We shall fix this! It is a little different from the book.
Symbol Tables

- Traditional method for managing binders in system.
- Logically one symbol table per scope.
  - Really a tree: common ancestor blocks can be shared.
  - Can get messy if hash-table is used.
- Interferes with semantic rules/actions.
- We shall fix this! It is a little different from the book.
Symbol Tables

- Traditional method for managing binders in system.
- Logically one symbol table per scope.
  - Really a tree: common ancestor blocks can be shared.
  - Can get messy if hash-table is used.
- Interferes with semantic rules/actions.
- *We shall fix this!* It is a little different from the book.
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:**
  \[ P \rightarrow S \]
  \[ S \rightarrow \text{int } V = E; S \mid \text{print } V; S \mid \epsilon \]
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:**
  \[
  P \rightarrow S \\
  S \rightarrow \text{int } V = E; S \mid \text{print } V; S \mid \epsilon
  \]
Environment Example

```
int = ⟨id, 2⟩
    ⟨num, 32⟩

float = ⟨id, 3⟩
    ⟨num, .8⟩

float = 
    ⟨id, 1⟩
    +
    ⟨id, 2⟩
    ∗
    ⟨id, 3⟩
    ⟨num, 60⟩
```
Environment Example

\[
\begin{align*}
\text{int} &= \langle \text{id}, 2 \rangle \\
\text{num} &= \langle \text{id}, 3 \rangle, \langle \text{num}, 32 \rangle \\
\text{float} &= \langle \text{id}, 3 \rangle, \langle \text{num}, 0.8 \rangle \\
\text{float} &= \langle \text{id}, 1 \rangle + \langle \text{id}, 2 \rangle \ast \langle \text{id}, 3 \rangle, \langle \text{num}, 60 \rangle
\end{align*}
\]
Environment Example

```
int= ⟨id, 2⟩
    |   
    v   
⟨num, 32⟩

float= ⟨id, 3⟩
    |   
    v   
⟨num, .8⟩

float= ⟨id, 1⟩
    |   
    v   
+   
    ↙   
    ↙   
    v   
⟨id, 2⟩

∗   
    ↙   
    ↙   
    v   
⟨id, 3⟩
    |   
    v   
⟨num, 60⟩
```

<table>
<thead>
<tr>
<th>id</th>
<th>lexeme</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>initial</td>
<td>int</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>lexeme</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>initial</td>
<td>int</td>
</tr>
<tr>
<td>3</td>
<td>rate</td>
<td>float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>lexeme</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>initial</td>
<td>int</td>
</tr>
<tr>
<td>3</td>
<td>rate</td>
<td>float</td>
</tr>
</tbody>
</table>

Eva Rose, Kristoffer Rose
Compiler Construction (CSCI-GA.2130-001) 5. Name Analysis
October 2, 2014
Environment Example

```plaintext
int=  
⟨id, 2⟩  
     |  
     ⟨num, 32⟩

float=  
⟨id, 3⟩
     |  
     ⟨num, .8⟩

float=  
 ⟨id, 1⟩  
        +  
        ⟨id, 2⟩  
        *  
        ⟨id, 3⟩

          +  
          ⟨num, 60⟩
```

<table>
<thead>
<tr>
<th>id</th>
<th>lexeme</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>initial</td>
<td>int</td>
</tr>
<tr>
<td>3</td>
<td>rate</td>
<td>float</td>
</tr>
<tr>
<td>1</td>
<td>position</td>
<td>float</td>
</tr>
</tbody>
</table>

```
Environment Example with *Stack*

```
int = (id, 2) (num, 32)
float = (id, 3) (num, .8)
float = (id, 1) + (id, 2) * (id, 3) (num, 60)
```

<table>
<thead>
<tr>
<th>id</th>
<th>lexeme</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>initial</td>
<td>int</td>
</tr>
<tr>
<td>3</td>
<td>rate</td>
<td>float</td>
</tr>
<tr>
<td>1</td>
<td>position</td>
<td>float</td>
</tr>
</tbody>
</table>
Environment Example with Maps

```
int = ⟨id, 2⟩,
    ⟨num, 32⟩,

float = ⟨id, 3⟩,
    ⟨num, .8⟩,

float = ⟨id, 1⟩,
    +,
    ⋅
    ⟨id, 2⟩,
    ⟨id, 3⟩,
    ⟨num, 60⟩,

e = {initial ↦ int}
    e = {rate ↦ float
        initial ↦ int
    }
    e = {position ↦ float
        rate ↦ float
        initial ↦ int
    }
```
Environment Example with *Stack of Maps*

```
int =
 ⟨id, 2⟩
 ⟨num, 32⟩

float =
 ⟨id, 3⟩
 ⟨num, .8⟩

float =
 ⟨id, 1⟩
 +
 ⟨id, 2⟩
 *
 ⟨id, 3⟩
 ⟨num, 60⟩
```

```
e = {initial \mapsto \text{int}}
```

```
e = {rate \mapsto \text{float}}
```

```
e = {position \mapsto \text{float}}
```
Binding Construct with Local Symbol Table = Environment

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

\[ x \quad \text{V} \quad S \]

\[ \text{int=} \]

\[ x \]
Binding Construct with Local Symbol Table = Environment II

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

\[ V_1 \]

\[ V_2 \]

\[ S \]

\[ e = \{y \mapsto \text{float}, x \mapsto \text{int}\} \]}
Shadowing

```c
int x = 32;
int y;
{
    float x = .8;
    float y = x + x * 8;
}
y = y + x;
```
Binding Construct with Local Symbol Table = Environment III
### Example

<table>
<thead>
<tr>
<th><strong>Production</strong></th>
<th><strong>Semantic Rules</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{id} := E_1; S_2$</td>
<td>$E_1.e = S.e; \quad S_2.e = \text{Extend}(S.e, \text{id}.sym, E_1.t)$</td>
</tr>
<tr>
<td>$\mid { S_1 } S_2$</td>
<td>$S_1.e = S.e; \quad S_2.e = S.e$</td>
</tr>
<tr>
<td>$\mid \epsilon$</td>
<td></td>
</tr>
<tr>
<td>$E \rightarrow E_1 + T_2$</td>
<td>$E_1.e = E.e; \quad T_2.e = E.e; \quad E.t = \text{Unif}(E_1.t, T_2.t)$</td>
</tr>
<tr>
<td>$\mid T_1$</td>
<td>$T_1.e = E.e; \quad E.t = T_1.t$</td>
</tr>
<tr>
<td>$T \rightarrow T_1 * F_2$</td>
<td>$T_1.e = T.e; \quad F_2.e = T.e; \quad T.t = \text{Unif}(T_1.t, F_2.t)$</td>
</tr>
<tr>
<td>$\mid F_1$</td>
<td>$F_1.e = T.e; \quad T.t = F_1.t$</td>
</tr>
<tr>
<td>$F \rightarrow \text{id}$</td>
<td>$F.t = \text{Lookup}(F.e, \text{id}.sym)$</td>
</tr>
<tr>
<td>$\mid \text{int}$</td>
<td>$E.t = \text{Int}$</td>
</tr>
<tr>
<td>$\mid \text{float}$</td>
<td>$E.t = \text{Float}$</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>PRODUCTION</th>
<th>SEMANTIC RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{id} := E_1; S_2$</td>
<td>$E_1.e = S.e; S_2.e = \text{Extend}(S.e, \text{id}.sym, E_1.t)$</td>
</tr>
<tr>
<td></td>
<td>$S_1.e = S.e; S_2.e = S.e$</td>
</tr>
<tr>
<td></td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>$E \rightarrow E_1 + T_2$</td>
<td>$E_1.e = E.e; T_2.e = E.e; E.t = \text{Unif}(E_1.t, T_2.t)$</td>
</tr>
<tr>
<td></td>
<td>$T_1.e = E.e; E.t = T_1.t$</td>
</tr>
<tr>
<td>$T \rightarrow T_1 * F_2$</td>
<td>$T_1.e = T.e; F_2.e = T.e; T.t = \text{Unif}(T_1.t, F_2.t)$</td>
</tr>
<tr>
<td></td>
<td>$F_1.e = T.e; T.t = F_1.t$</td>
</tr>
<tr>
<td>$F \rightarrow \text{id}$</td>
<td>$F.t = \text{Lookup}(F.e, \text{id}.sym)$</td>
</tr>
<tr>
<td></td>
<td>$\text{int}$</td>
</tr>
<tr>
<td></td>
<td>$E.t = \text{Int}$</td>
</tr>
<tr>
<td></td>
<td>$\text{float}$</td>
</tr>
<tr>
<td></td>
<td>$E.t = \text{Float}$</td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{id} := E_1; S_2 \quad</td>
<td>\quad E_1.e = S.e; \quad S_2.e = \text{Extend}(S.e, \text{id}.\text{sym}, E_1.t)$</td>
</tr>
<tr>
<td>$\mid {\ S_1 \ } \ S_2 \quad</td>
<td>\quad S_1.e = S.e; \quad S_2.e = S.e$</td>
</tr>
<tr>
<td>$\mid \epsilon \quad</td>
<td>\quad E_1.e = E.e; \quad T_2.e = E.e; \quad E.t = \text{Unif}(E_1.t, T_2.t)$</td>
</tr>
<tr>
<td>$E \rightarrow E_1 + T_2 \quad</td>
<td>\quad T_1.e = E.e; \quad E.t = T_1.t$</td>
</tr>
<tr>
<td>$\mid T_1 \quad</td>
<td>\quad T_1.e = T.e; \quad F_2.e = T.e; \quad T.t = \text{Unif}(T_1.t, F_2.t)$</td>
</tr>
<tr>
<td>$T \rightarrow T_1 \ast F_2 \quad</td>
<td>\quad F_1.e = T.e; \quad T.t = F_1.t$</td>
</tr>
<tr>
<td>$\mid F_1 \quad</td>
<td>\quad F.t = \text{Lookup}(F.e, \text{id}.\text{sym})$</td>
</tr>
<tr>
<td>$F \rightarrow \text{id} \quad</td>
<td>\quad F.t = \text{Int}$</td>
</tr>
<tr>
<td>$\mid \text{int} \quad</td>
<td>\quad E.t = \text{Int}$</td>
</tr>
<tr>
<td>$\mid \text{float} \quad</td>
<td>\quad E.t = \text{Float}$</td>
</tr>
</tbody>
</table>
Introduction

Programming Language Basics

Symbol Tables = Environments

HACS

Project Milestone 1
Binding Construct

\[ \text{int} = \{ x \mapsto \text{int}, \ldots \} \]

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

\[
\begin{align*}
\text{int} &= [x] \quad V_1 \\
\text{float} &= [y] \quad V_2 \\
S &\quad e = \{x \mapsto \text{int}, \ldots\} \\
S &\quad e = \{y \mapsto \text{float}, x \mapsto \text{int}, \ldots\}
\end{align*}
\]
Binding Construct à la HACS

\[
\begin{align*}
&\text{int} = V_1 \quad \text{float} = V_2 \\
&[x] \quad [x] \\
&e = \{x \mapsto \text{int}, \ldots \} \\
&S \\
& e = \{x \mapsto \text{float}, \ldots \}
\end{align*}
\]
HACS is Higher-order Attribute Contraction Schemes II

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

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

\[ \text{sort } P \mid \langle \langle S\rangle \rangle \; ; \]

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

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

\[ \text{sort } V | \text{symbol } [\langle ID \rangle] ; \]

\[ \text{sort } P | [\langle S \rangle] ; \]

\[ \text{sort } S | [\text{int } \langle V \text{ binds } x \rangle = \langle E \rangle; \quad \langle S[x \text{ as } V] \rangle ] \]
\[ | [\text{print } \langle V \rangle; \quad \langle S \rangle ] \]
\[ | [ ] ; \]
Project Milestone 1 Issues

- Lexer issues: ., comments, character classes.
- Spaces in sort $[\_\_]-$declarations.
- Document, document, document...
  - Choices where specification is imprecise.
  - Who you discussed strategy with.
  - How to test your code and what is tested.
Questions?

evarose@cs.nyu.edu  krisrose@cs.nyu.edu