Name Analysis

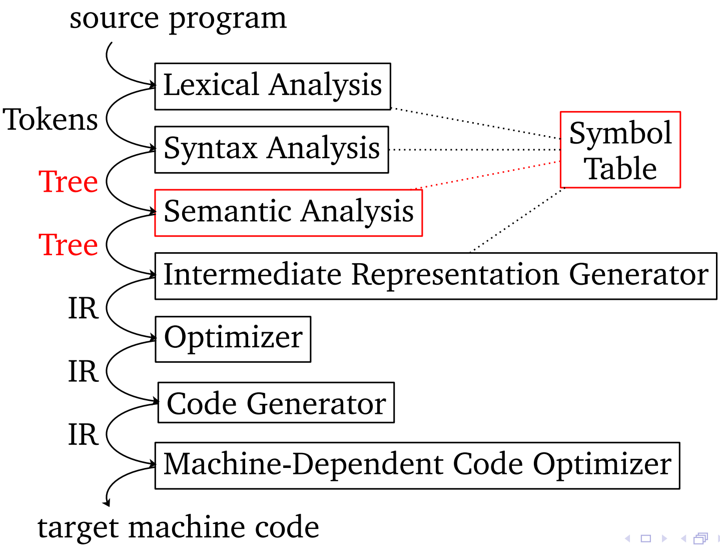
Kristoffer H. Rose Eva Rose
{krisrose,evarose}@cs.nyu.edu

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

March 3, 2014

Outline

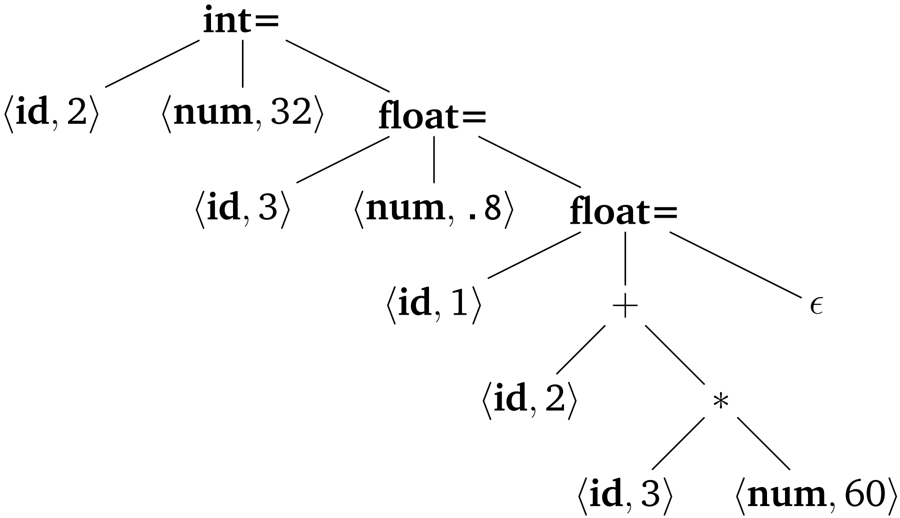
Context


source program
Lexical Analysis
Tokens
Syntax Analysis
Tree
Semantic Analysis
Tree
Intermediate Representation Generator
IR
Optimizer
IR
Code Generator
IR
Machine-Dependent Code Optimizer
target machine code
Symbol Table

Example Code

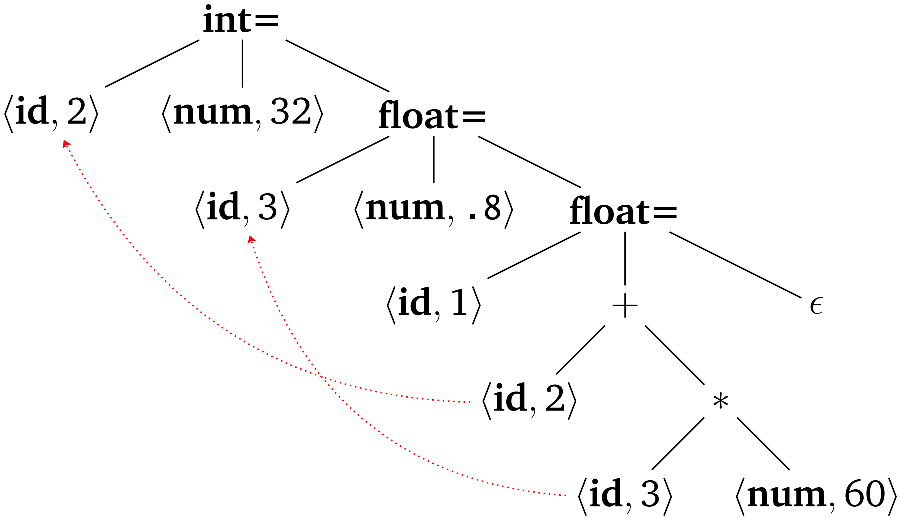
```
int initial = 32;
float rate = .8;
float position = initial + rate * 8;
```

Example Abstract Syntax Tree (AST)



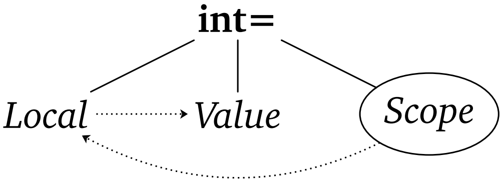
id	*lexeme*
1	`position`
2	`initial`
3	`rate`

Example Abstract Syntax Tree (AST) + “def-use”

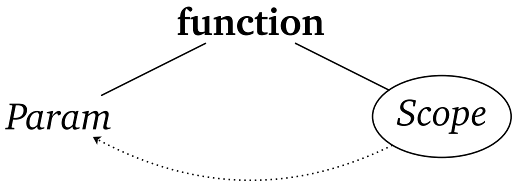


id	*lexeme*
1	`position`
2	`initial`
3	`rate`

Binding Construct



Binding Construct II



Outline

Compile time *vs* Runtime Values

$$\begin{tikzcd} \textit{Variable Use} \arrow[r, bend right, "\text{Compile-time Environment}"'] & \textit{Variable Definition} \arrow[r, bend right, "\text{Run-time Stack/Heap}"'] & \textit{Memory Location} \end{tikzcd}$$

Shadowing

```
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!*

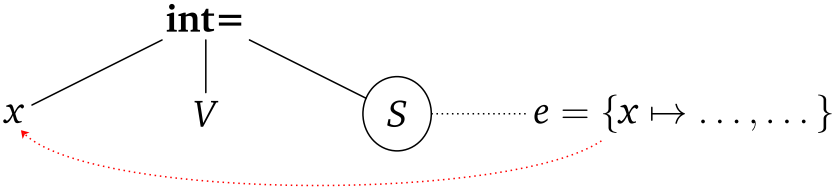
Symbol Tables

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

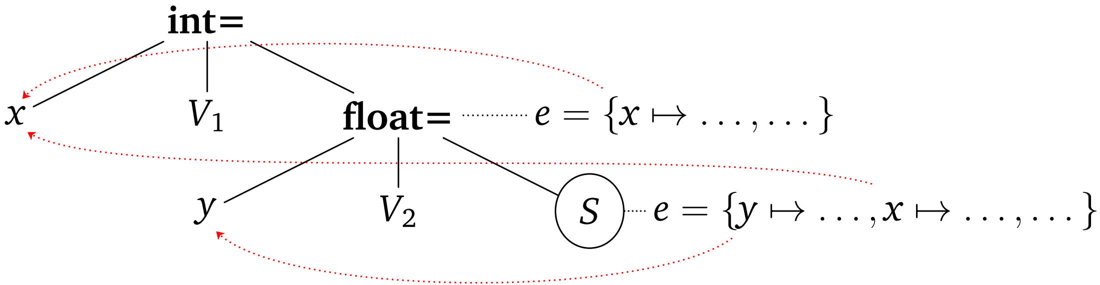
Symbol Tables

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

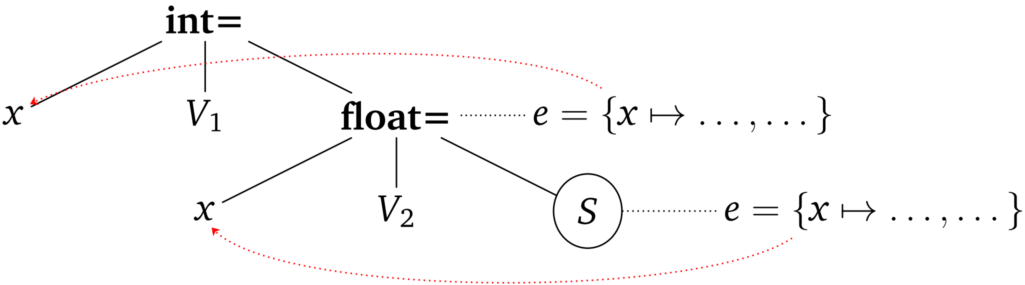
Binding Construct *with Local Symbol Table = Environment*



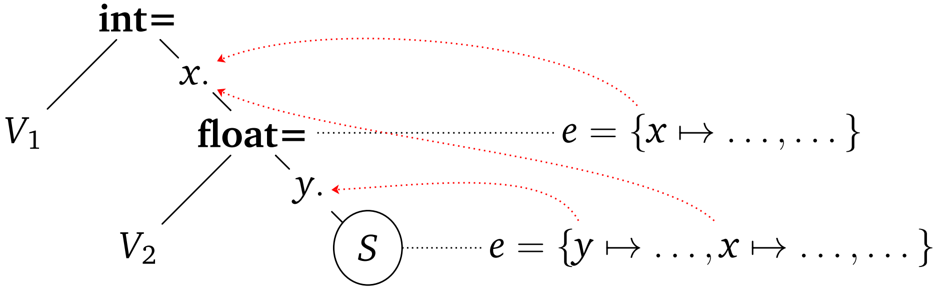
Binding Construct *with Local Symbol Table = Environment* II



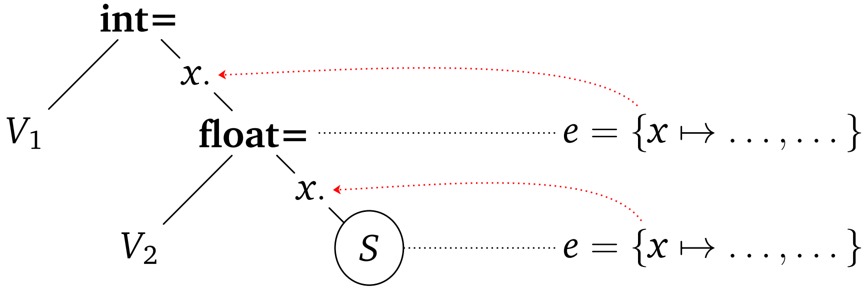
Binding Construct *with Local Symbol Table = Environment* III



Binding Construct *à la HACS*



Binding Construct *à la HACS*



HACS is *Higher-order* Attribute Contraction Schemes

- Traditional:

$$P \to S^*$$
$$S \to \textbf{int}\ V = E; \mid \textbf{print}\ V;$$

- Combine Scoping and Grammar:

$$P \to S$$
$$S \to \textbf{int}\ V = E;\ S \mid \textbf{print}\ V;\ S \mid \epsilon$$

HACS is *Higher-order* Attribute Contraction Schemes

- Traditional:

$$P \to S^*$$
$$S \to \textbf{int}\ V = E; \mid \textbf{print}\ V;$$

- Combine Scoping and Grammar:

$$P \to S$$
$$S \to \textbf{int}\ V = E;\ S \mid \textbf{print}\ V;\ S \mid \epsilon$$

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

$$P \to S$$

$$S \to \mathbf{int}\ V_{x.} \ \mathbf{=}\ E;\ S^{x} \mid \mathbf{print}\ V;\ S \mid \epsilon$$

$$\textbf{sort } V \quad \mid \ \textbf{symbol } \llbracket \langle ID \rangle \rrbracket \ ;$$

$$\textbf{sort } P \quad \mid \ \llbracket \langle S \rangle \rrbracket \ ;$$

$$\begin{aligned} \textbf{sort } S \quad & \mid \ \llbracket\ int\ \langle x{:}V \rangle = \langle E \rangle;\ \langle S[x{:}V] \rangle\ \rrbracket \\ & \mid \ \llbracket\ print\ \langle V \rangle;\ \langle S \rangle\ \rrbracket \\ & \mid \ \llbracket\rrbracket \ ; \end{aligned}$$

HACS is *Higher-order* Attribute Contraction Schemes II

$$P \to S$$

$$S \to \textbf{int}\ V_{x.}\ \textbf{=}\ E;\ S^{x} \mid \textbf{print}\ V;\ S \mid \epsilon$$

$$\textbf{sort}\ V \quad \mid\ \textbf{symbol}\ [\![\langle ID \rangle]\!]\ ;$$

$$\textbf{sort}\ P \quad \mid\ [\![\langle S \rangle]\!]\ ;$$

$$\begin{aligned} \textbf{sort}\ S \quad & \mid\ [\![\ int\ \langle x{:}V \rangle\ =\ \langle E \rangle;\ \langle S[x{:}V] \rangle\ ]\!] \\ & \mid\ [\![\ print\ \langle V \rangle;\ \langle S \rangle\ ]\!] \\ & \mid\ [\![]\!]\ ; \end{aligned}$$

Example

Production	Semantic Rules
$S \rightarrow \textbf{id} := E$	$E.e = S.e; S.sym = \textbf{id}.sym; S.t = E.t$
$\mid \{ \ S^* \ \}$	$S^*.e = S.e; S.sym = \epsilon; S.t = \epsilon$
$S^* \rightarrow S_1 \ S_2^*$	$S_1.e = S^*.e$ $S_2^*.e = \text{if } S_1.sym \neq \epsilon \text{ then extend}(S^*.e, S_1.sym, S_1.t) \text{ el}$
$\mid \epsilon$	

Outline

Transition

. . . On To The Blackboard!

The following two slides have a sanitized version of the file we wrote together in class.

Extending hw4 Question 2.2 with Declarations (I)

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

token *ID* | [*a*−*z*]+ ('_' [0−9]+)∗ ;
sort *V* | **symbol** $[\![\langle ID\rangle]\!]$;

sort *YesNo* | *Yes* | *No* ;
attribute $\uparrow z(YesNo)$;
attribute $\downarrow e\{V{:}YesNo\}$;

sort *E* | $[\![$ (*let* $\langle[x{:}V]\rangle$ $\langle E\rangle$ $\langle E[x{:}V]\rangle$) $]\!]$ | $[\![$ $\langle V\rangle$ $]\!]$
| $\uparrow z$ | **scheme** $Z(E){\downarrow}e$ | **scheme** $Z2(E){\downarrow}e$;

Extending hw4 Question 2.2 with Declarations (II)

$$Z(\llbracket\ (\ \mathit{let}\ \ x\ \langle E\#2\rangle\ \langle E\#3[x]\rangle\ )\ \rrbracket)$$
$$\quad\to Z2(\llbracket\ (\ \mathit{let}\ \ x\ \langle E\ Z(\#2)\rangle\ \langle E\#3[x]\rangle\ )\ \rrbracket)\ ;$$

$$Z2(\llbracket\ (\ \mathit{let}\ \ x\ \langle E\#2\uparrow z(\#z2)\rangle\ \langle E\#3[x]\rangle\ )\ \rrbracket)$$
$$\quad\to \llbracket\ (\ \mathit{let}\ \ x\ \langle E\#2\rangle\ \langle E\ Z(E\#3[x])\downarrow e\{x{:}\#z2\}\rangle\ )\ \rrbracket;$$

$$\llbracket\ (\ \mathit{let}\ \ x\ \langle E\#2\rangle\ \langle E\#3[x]\uparrow z(\#z3)\rangle\ )\ \rrbracket\uparrow z(\#z3)\ ;$$

$$Z(\llbracket\ x\ \rrbracket\ )\downarrow e\{x{:}\#z\}\to\llbracket x\ \rrbracket\uparrow z(\#z)\ ;$$
$$\}$$

Note: Notation still subject to change.

Questions?

krisrose@cs.nyu.edu