Syntax-Directed Translation

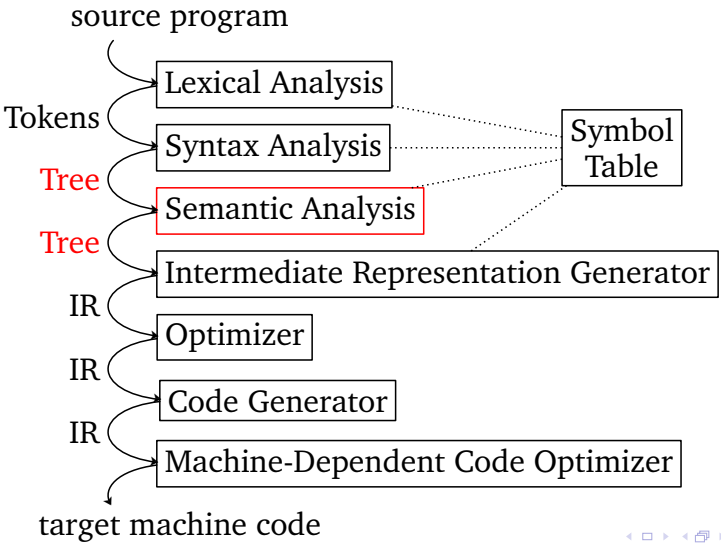
Kristoffer H. Rose Eva Rose
{*krisrose,evarose*}*@cs.nyu.edu*

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

February 24, 2014

Outline

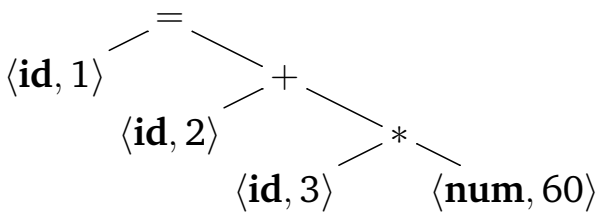
Context


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

From Abstract Syntax Tree (AST)

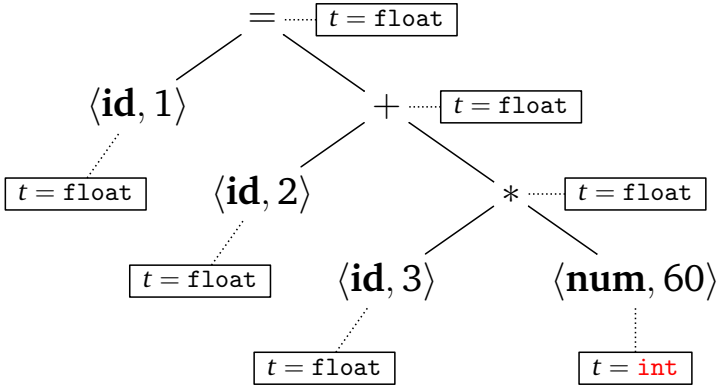
```
position = initial + rate * 60
```

parsed into *abstract syntax tree*:



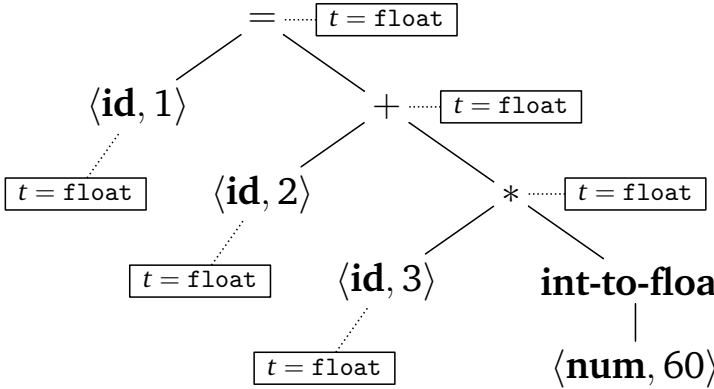
id	*lexeme*
1	`position`
2	`initial`
3	`rate`

. . . to Annotated AST. . .



id	*lexeme*	*t*
1	position	float
2	initial	float
3	rate	float

. . . to "Fixed" AST



id	*lexeme*	*t*
1	position	float
2	initial	float
3	rate	float

Outline

Goal

From

$$expr \rightarrow expr_1 + term_2$$

to

$$\text{Code for } expr \text{ to } R = \begin{cases} \text{Code for } expr_1 \text{ to } R_1 \\ \text{Code for } term_2 \text{ to } R_2 \\ \text{Code for } R := R_1 + R_2 \end{cases}$$

Goal

From

$$expr \rightarrow expr_1 + term_2$$

to

$$\boxed{\text{Code for } expr \text{ to } R} = \begin{cases} \boxed{\text{Code for } expr_1 \text{ to } R_1} \\ \boxed{\text{Code for } term_2 \text{ to } R_2} \\ \boxed{\text{Code for } R := R_1 + R_2} \end{cases}$$

Syntax-Directed Definition

Define attributes:

c : code generated from non-terminal

r : register used for result of computing non-terminal

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1 + T_2$	$E_1.r = fresh; T_2.r = fresh$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"

Syntax-Directed Definition

Define attributes:

c : code generated from non-terminal

r : register used for result of computing non-terminal

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1 + T_2$	$E_1.r = fresh; T_2.r = fresh$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"

Attributes

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1 + T_2$	$E_1.r = \mathit{fresh}\text{-}R; T_2.r = \mathit{fresh}\text{-}R$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"

c : Set for *entire production* — synthesized

r : Set for *sub-production* — inherited

Attributes

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1 + T_2$	$E_1.r = \mathit{fresh}\text{-}R; T_2.r = \mathit{fresh}\text{-}R$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"

c : Set for *entire production* $\rightarrow$ synthesized

r : Set for *sub-production* $\rightarrow$ inherited

Attributes

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"

c : Set for *entire production* $\rightarrow$ synthesized

r : Set for *sub-production* $\rightarrow$ inherited

Example

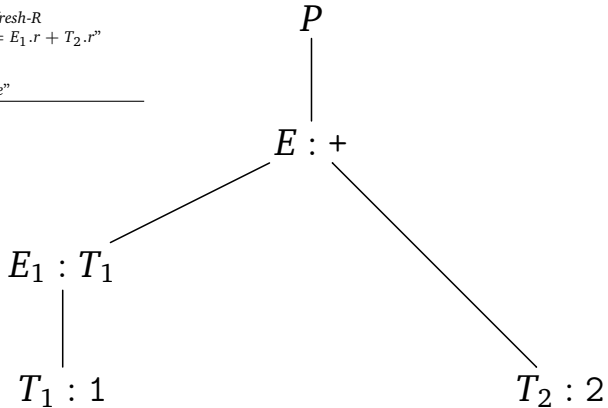
PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \mathit{fresh\text{-}R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \mathit{fresh\text{-}R}; T_2.r = \mathit{fresh\text{-}R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"

Example (text)

1+2

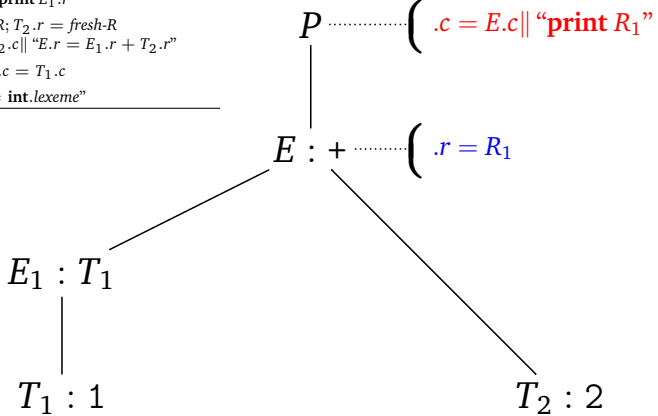
Example (AST)

Production	Semantic Rules
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



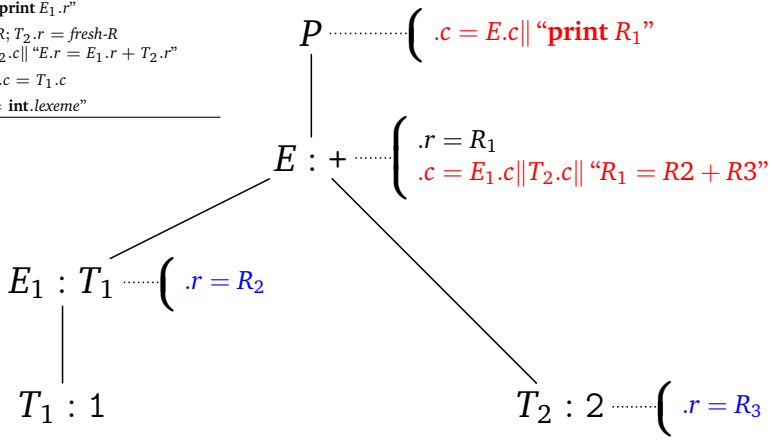
Example (attributes I)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



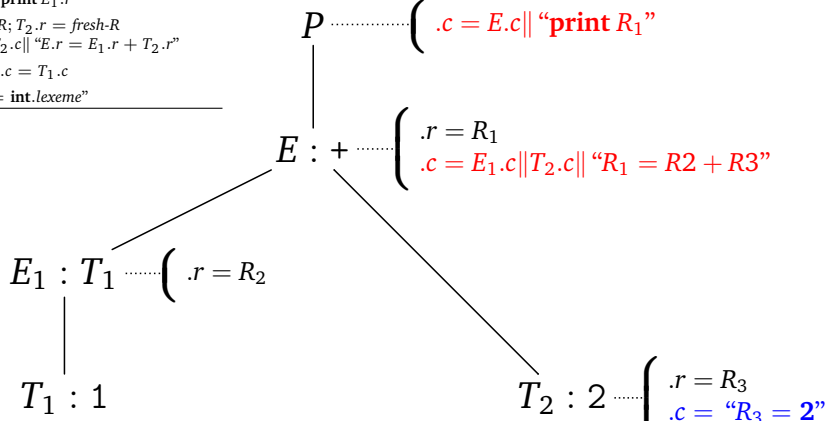
Example (attributes II)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



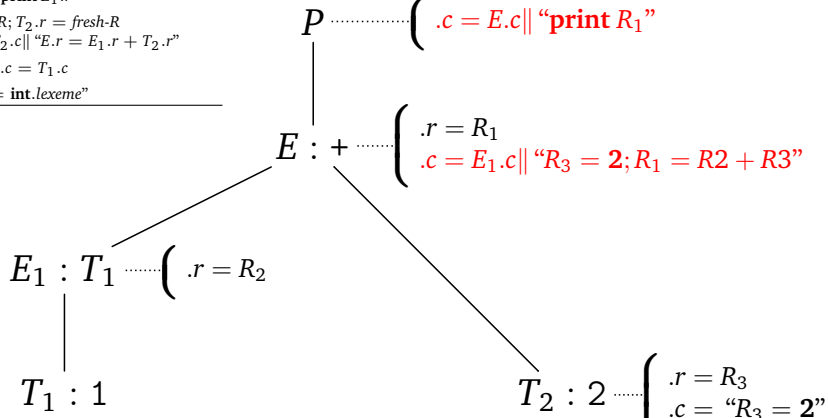
Example (attributes III)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



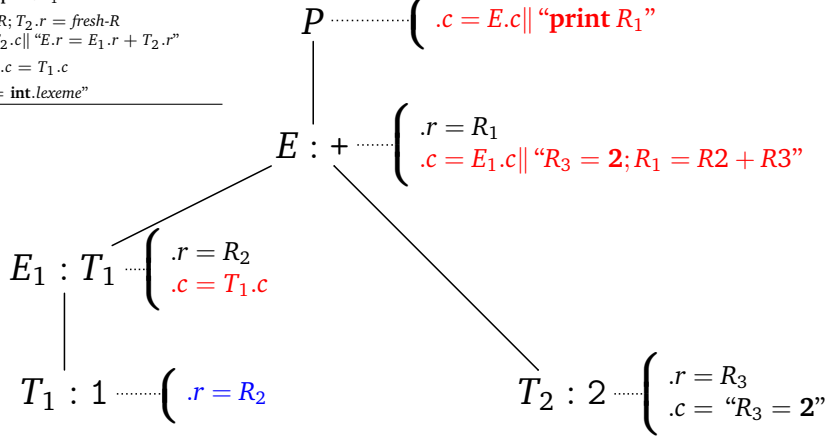
Example (attributes IV)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



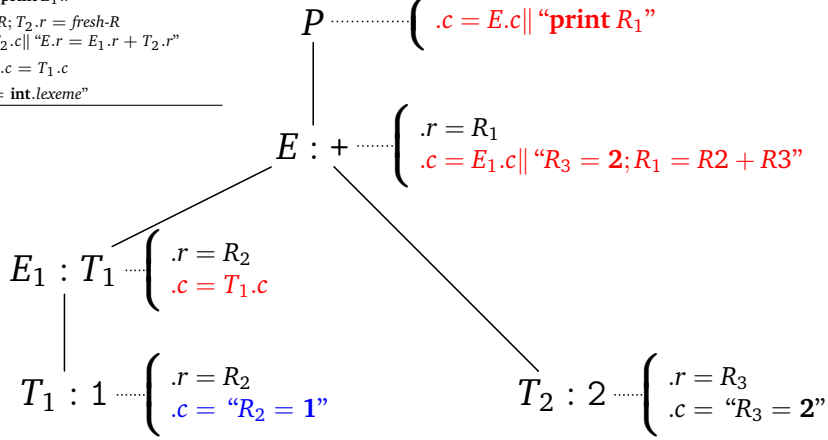
Example (attributes V)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



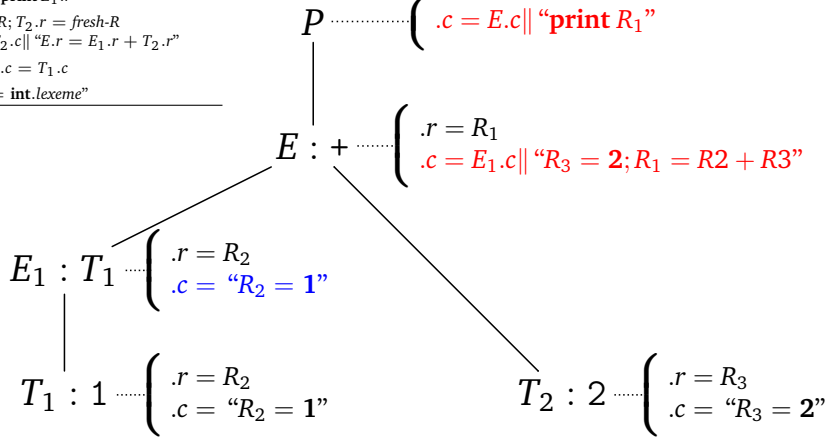
Example (attributes VI)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



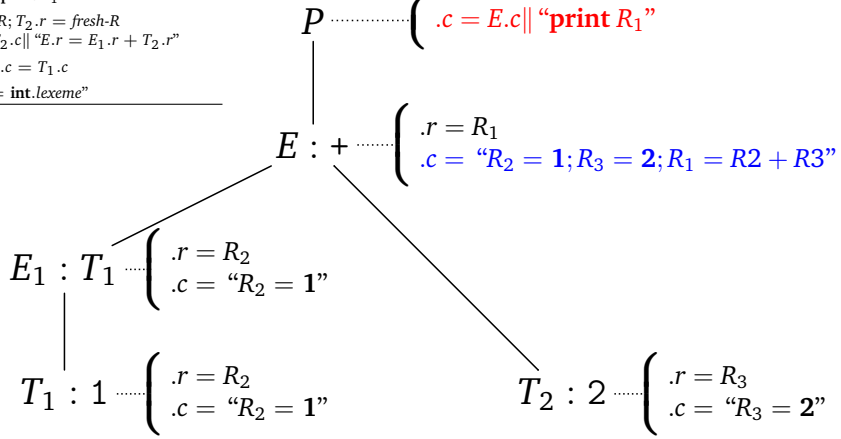
Example (attributes VII)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



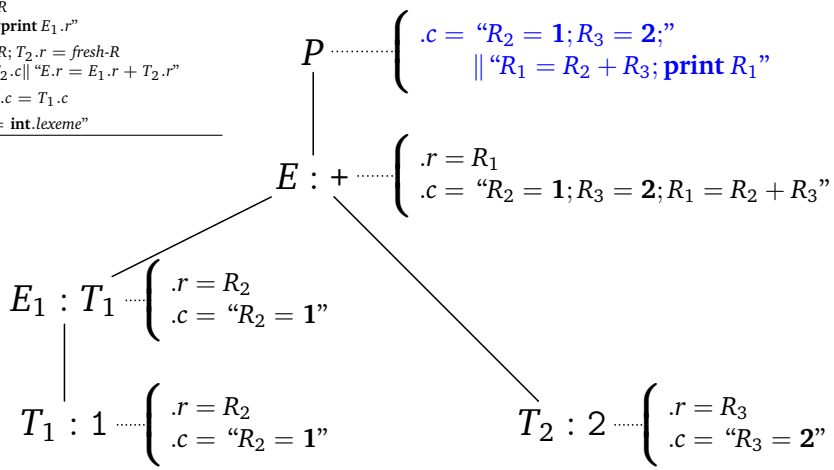
Example (attributes VIII)

PRODUCTION	SEMANTIC RULES
$P \to E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \to E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \to T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \to$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"

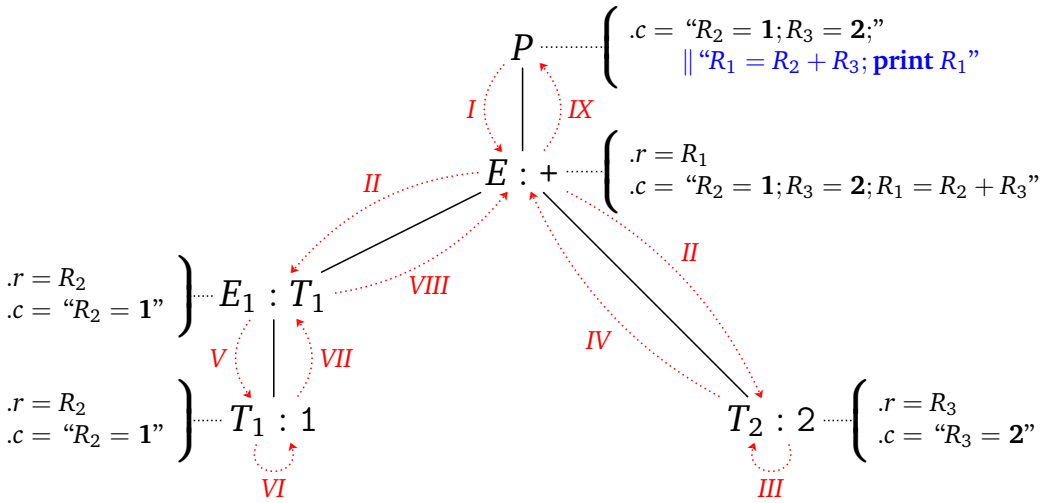


Example (attributes IX)

PRODUCTION	SEMANTIC RULES
$P \to E_1$	$E_1.r = \textit{fresh-R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \to E_1 + T_2$	$E_1.r = \textit{fresh-R}; T_2.r = \textit{fresh-R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \to T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \to$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"



Example (attribute evaluation order)



Semantic Actions

- if we know the tree traversal mechanism left to right post-order—
- then we can replace concatenations with appending side effects.

Semantic Actions

- if we know the tree traversal mechanism left to right post-order—
- then we can replace concatenations with appending side effects.

Semantic Actions (II)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \mathit{fresh\text{-}R}$ $P.c = E_1.c \Vert$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \mathit{fresh\text{-}R}; T_2.r = \mathit{fresh\text{-}R}$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"

Semantic Actions (III)

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \mathit{fresh}\text{-}R$ $\{$ **emit** "**print** $E_1.r$" $\}$
$E \rightarrow E_1 + T_2$	$E_1.r = \mathit{fresh}\text{-}R; T_2.r = \mathit{fresh}\text{-}R$ $\{$ **emit** "$E.r = E_1.r + T_2.r$" $\}$
$E \rightarrow T_1$	$T_1.r = E.r$
$T \rightarrow$ **int**	$\{$ **emit** "$T.r =$ **int**.*lexeme*" $\}$

Outline

SDD = Grammar + Semantic (Attribute) Rules

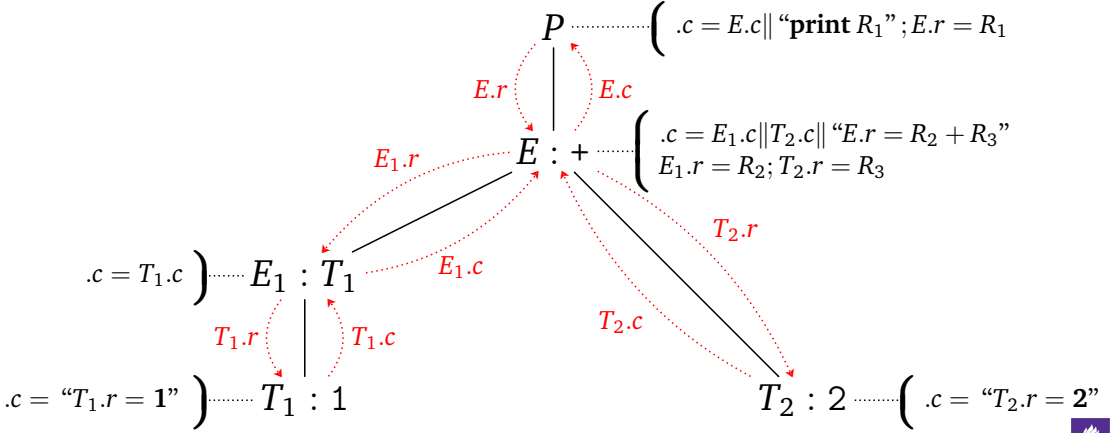
Synthesized attribute: Defined for result non-terminal of production.

Inherited attribute: Defined for (or "sent to") component non-terminals of production.

Cycles possible!

PRODUCTION	SEMANTIC RULES
$A \rightarrow B_1$	$A.s = B.i \; ; \; B.i = A.s + 1$

Dependency Graph



Good SDD Classes

S-attributed: Only synthesized attributes.

L-attributed: Each attribute is *either*

- synthesized (without restrictions), or
- inherited but for a production

$$A \to X_1 X_2 \ldots X_n$$

where inherited attributes for X_j only depends on inherited attributes for A and $X_1, \ldots, X_{j-1}$ and on synthesized attributes of $X_1, \ldots, X_{j-1}$.

Good SDD Classes

S-attributed: Only synthesized attributes.

L-attributed: Each attribute is *either*

- synthesized (without restrictions), *or*
- inherited but for a production

$$A \to X_1 X_2 \ldots X_n$$

where inherited attributes for X_i *only* depends on inherited attributes for A and $X_1, \ldots, X_{i-1}$ and on synthesized attributes of $X_1, \ldots, X_{i-1}$.

Good SDD Classes

S-attributed: Only synthesized attributes.

L-attributed: Each attribute is *either*

- synthesized (without restrictions), *or*
- inherited but for a production

$$A \to X_1 X_2 \ldots X_n$$

where inherited attributes for X_i *only* depends on inherited attributes for A and $X_1, \ldots, X_{i-1}$ and on synthesized attributes of $X_1, \ldots, X_{i-1}$.

Good SDD Classes

S-attributed: Only synthesized attributes.

L-attributed: Each attribute is *either*

- synthesized (without restrictions), *or*
- inherited but for a production

$$A \to X_1 X_2 \ \ldots \ X_n$$

where inherited attributes for X_i *only* depends on inherited attributes for A and $X_1, \ldots, X_{i-1}$ and on synthesized attributes of $X_1, \ldots, X_{i-1}$.

Outline

HACS-Attributed

HACS supports:

- S-attributed SDDs.
- Recursive Translation Schemes with named arguments.

For convenience the named arguments are called inherited attributes.

HACS-Attributed

HACS supports:

- S-attributed SDDs.
- Recursive Translation Schemes with named arguments.

For convenience the named arguments are called inherited attributes.

Example, Again

PRODUCTION	SEMANTIC RULES
$P \rightarrow E_1$	$E_1.r = \mathit{fresh\text{-}R}$ $P.c = E_1.c \,\|\|$ "**print** $E_1.r$"
$E \rightarrow E_1 + T_2$	$E_1.r = \mathit{fresh\text{-}R}; T_2.r = \mathit{fresh\text{-}R}$ $E.c = E_1.c \,\|\| T_2.c \,\|\|$ "$E.r = E_1.r + T_2.r$"
$E \rightarrow T_1$	$T_1.r = E.r; E.c = T_1.c$
$T \rightarrow$ **int**	$T.c =$ "$T.r =$ **int**.*lexeme*"

Grammar

$$P \to E_1 \qquad E \to E_1 + T_2 \qquad E \to T_1 \qquad T \to \textbf{int}$$

token *INTEGER* | [0−9]+ ;

sort P | ⟦ ⟨E⟩ ⟧ ;
sort E | ⟦ ⟨E⟩ + ⟨T⟩ ⟧ | ⟦ ⟨T⟩ ⟧ ;
sort T | ⟦ ⟨*INTEGER*⟩ ⟧;

Grammar

$$P \to E_1 \qquad E \to E_1 + T_2 \qquad E \to T_1 \qquad T \to \textbf{int}$$

token *INTEGER* | [0−9]+ ;

sort P | ⟦ ⟨E⟩ ⟧ ;
sort E | ⟦ ⟨E⟩ + ⟨T⟩ ⟧ | ⟦ ⟨T⟩ ⟧ ;
sort T | ⟦ ⟨*INTEGER*⟩ ⟧;

Attribute Sorts

token *REGISTER* | `'R'` (`'_'` ⟨*INTEGER*⟩)? ;

sort *Reg* | **symbol** ⟦⟨*REGISTER*⟩ ⟧;
sort *Code* | ⟦ *print* ⟨*Reg*⟩ ⟧@*1*
| ⟦ ⟨*Reg*⟩ = ⟨*Reg*⟩ + ⟨*Reg*⟩ ⟧@*1*
| ⟦ ⟨*Reg*⟩ = ⟨*INTEGER*⟩ ⟧@*1*
| ⟦ ⟨*Code*@*1*⟩ ;¶ ⟨*Code*⟩ ⟧
| ⟦⟧ ;

attribute ↓*r*(*Reg*) ;
attribute ↑*c*(*Code*) ;

Example, Again

$P \rightarrow E_1$	$E_1.r = \mathit{fresh}\text{-}R \quad P.c = E_1.c \Vert$ “**print** $E_1.r$”

Example, Again

$P \to E_1$	$E_1.r = \mathit{fresh}\text{-}R \quad P.c = E_1.c \Vert$ "**print** $E_1.r$"

sort P | **scheme** $R\_P(P)$ | **scheme** $R\_P1(P, Reg)$;

$R\_P(\llbracket\ \langle E\#1 \rangle\ \rrbracket)$
$\quad \to R\_P1(\llbracket\ \langle E\ R\_E(E\#1)\ \downarrow r(\llbracket R\_1 \rrbracket) \rangle\ \rrbracket, \llbracket R\_1 \rrbracket)$;

$R\_P1(\llbracket\ \langle E\#1\ \uparrow c(Code\#c1) \rangle\ \rrbracket, Reg\#r)$
$\quad \to \llbracket\ \langle E\#1 \rangle\ \rrbracket \uparrow c(\llbracket\ \langle Code\#c1 \rangle\ ;\ print\ \langle Reg\#r \rangle\ \rrbracket)$;

Example, Again

$E \to T_1$	$T_1.r = E.r; E.c = T_1.c$

Example, Again

$E \to T_1$	$T_1.r = E.r; E.c = T_1.c$

sort E | **scheme** $R\_E(E) \downarrow r$;

$R\_E(\llbracket \langle T\#1 \rangle \rrbracket) \downarrow r(Reg\#r)$
$\quad \to R\_E1(\llbracket \langle T\ R\_T(T\#1) \downarrow r(Reg\#r) \rangle \rrbracket)$;

| **scheme** $R\_E1(E)$;

$R\_E1(\llbracket \langle T\#1 \uparrow c(Code\#c1) \rangle \rrbracket)$
$\quad \to \llbracket \langle T\#1 \rangle \rrbracket \uparrow c(Code\#c1)$;

Example, Again

$E \rightarrow E_1 + T_2$	$E_1.r = \mathit{fresh}\text{-}R; T_2.r = \mathit{fresh}\text{-}R$ $E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"

$R\_E(\llbracket \langle E\#1 \rangle + \langle T\#2 \rangle \rrbracket) \downarrow r(Reg\#r)$
$\rightarrow R\_E2(\llbracket \langle E\ R\_E(E\#1) \downarrow r(\llbracket R\_1 \rrbracket) \rangle$
$+ \langle T\ R\_T(T\#2) \downarrow r(\llbracket R\_2 \rrbracket) \rangle \rrbracket,$
$Reg\#r, \llbracket R\_1 \rrbracket, \llbracket R\_2 \rrbracket)$;

| **scheme** $R\_E2(E, Reg, Reg, Reg)$;

$R\_E2(\llbracket \langle E\#1 \uparrow c(Code\#c1) \rangle + \langle T\#2 \uparrow c(Code\#c2) \rangle \rrbracket, Reg\#r, Reg\#r1, Reg\#r2)$
$\rightarrow \llbracket \langle E\#1 \rangle + \langle T\#2 \rangle \rrbracket$
$\uparrow c(\llbracket \langle Code\#c1 \rangle ; \langle Code\#c2 \rangle ; \langle Reg\#r \rangle = \langle Reg\#r1 \rangle + \langle Reg\#r2 \rangle \rrbracket)$;

Example, Again

$E \to E_1 + T_2$	$E_1.r = \mathit{fresh\text{-}R}; T_2.r = \mathit{fresh\text{-}R}$
	$E.c = E_1.c \Vert T_2.c \Vert$ "$E.r = E_1.r + T_2.r$"

$$
\begin{aligned}
&R\_E([\![\ \langle E\#1\rangle + \langle T\#2\rangle\ ]\!])\downarrow r(\mathit{Reg}\#r)\\
&\quad \to R\_E2([\![\ \langle E\ R\_E(E\#1)\downarrow r([\![R\_1]\!])\rangle\\
&\qquad\qquad + \langle T\ R\_T(T\#2)\downarrow r([\![R\_2]\!])\rangle\ ]\!],\\
&\qquad\quad \mathit{Reg}\#r, [\![R\_1]\!], [\![R\_2]\!])\ ;
\end{aligned}
$$

$$
\mid \textbf{scheme}\ R\_E2(E, \mathit{Reg}, \mathit{Reg}, \mathit{Reg})\ ;
$$

$$
\begin{aligned}
&R\_E2([\![\ \langle E\#1\uparrow c(\mathit{Code}\#c1)\rangle + \langle T\#2\uparrow c(\mathit{Code}\#c2)\rangle\ ]\!], \mathit{Reg}\#r, \mathit{Reg}\#r1, \mathit{Reg}\#r2)\\
&\quad \to [\![\ \langle E\#1\rangle + \langle T\#2\rangle\ ]\!]\\
&\qquad \uparrow c([\![\ \langle \mathit{Code}\#c1\rangle\ ;\ \langle \mathit{Code}\#c2\rangle\ ;\ \langle \mathit{Reg}\#r\rangle = \langle \mathit{Reg}\#r1\rangle + \langle \mathit{Reg}\#r2\rangle\ ]\!])\ ;
\end{aligned}
$$

Example, Again

$T \rightarrow \mathbf{int}$	$T.c = \text{“}T.r = \mathbf{int}.lexeme\text{”}$

sort T | **scheme** $R\_T(T) \downarrow r$;

$R\_T(\llbracket \langle INTEGER\#lexeme \rangle \rrbracket) \downarrow r(Reg\#r)$
$\rightarrow \llbracket \langle INTEGER\#lexeme \rangle \rrbracket \uparrow c(\llbracket \langle Reg\#r \rangle = \langle INTEGER\#lexeme \rangle \rrbracket)$;

Example, Again

$T \rightarrow \mathbf{int}$	$T.c = \text{“}T.r = \mathbf{int}.lexeme\text{”}$

sort T | **scheme** $R\_T(T) \downarrow r$;

$R\_T(\llbracket \langle INTEGER\#lexeme \rangle \rrbracket) \downarrow r(Reg\#r)$
$\quad \rightarrow \llbracket \langle INTEGER\#lexeme \rangle \rrbracket \uparrow c(\llbracket \langle Reg\#r \rangle = \langle INTEGER\#lexeme \rangle \rrbracket)$;

Questions?

krisrose@cs.nyu.edu