Week 5: Book

Tuesday, February 15, 2022 10:17 AM

2.3 Syntax Directed Translation

Syntax directed translation: attach rules or program fragments to productions in a grammar.

↳ ex: consider expr generated by the production

$$\underbrace{expr \rightarrow expr_1}_{} + term$$

the $expr_1$ is used to distinguish the head expr from the body expr.

Translate expr into pseudo-code:

```
translate expr1;
translate term;
handle +;
```

The example in this section translates from infix to postfix notation.

Two concepts related to syntax directed translation:

1. Attribute: it is any quantity associated with a programming construct. Examples are data types, number of instructions, location of 1st instruction, etc

↳ Since we use grammar symbols (NONTERMINAL / TERMINAL) for constructs; attributes from constructs are the symbols that represent constructs.

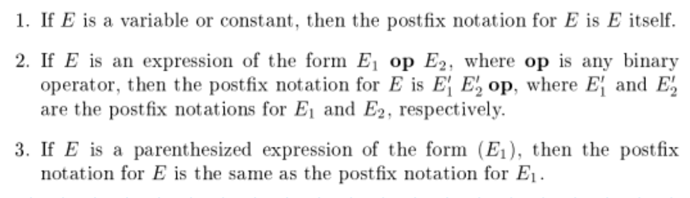
2. (Syntax-directed) translation schemes: it is a notation for attaching program fragments to productions.

↳ are executed when the production is used during syntax analysis

Syntax directed translation is used here to translate infix to post fix, evaluate expressions, build syntax trees for constructs.

2.3.1 Postfix Notation

The post fix notation for an expression E can be defined:



1. If E is a variable or constant, then the postfix notation for E is E itself.
2. If E is an expression of the form E_1 **op** E_2, where **op** is any binary operator, then the postfix notation for E is $E'_1\ E'_2$ **op**, where E'_1 and E'_2 are the postfix notations for E_1 and E_2, respectively.
3. If E is a parenthesized expression of the form (E_1), then the postfix notation for E is the same as the postfix notation for E_1.

Example: the postfix notation for $(9-5)+2$ is $95-2+$

↳ The translation of 9, 5and 2 are themselves (rule 1).

↳ The translation of $(9-5)$ is $95-$ by rule 2 and rule 3.

↳ after the translation of the parentheses expressions, we apply rule 2 to the whole.

↳ $\left.\begin{array}{l} E_1 \rightarrow (9-5) \\ E_2 \rightarrow 2 \end{array}\right\} \Rightarrow \boxed{95-2+}$ ✓

Example 2: $9-(5+2)$

↳ $52+$

↳ $\left.\begin{array}{l} E_1 = 9 \\ E_2 = (5+2) \end{array}\right\} \Rightarrow \boxed{952+-}$ ✓

→ Note: No parenthesis are needed in postfix, since the position and arity (# of args) allows only one decoding of a postfix expr.

↳ the trick: scan from L → R until you find an operator
↳ look left of operator for their number of operands
↳ group this operator with its operands
↳ evaluate this and replace on the postfix w/ the result
↳ repeat the process.

Example 3: the postfix 952+-3*

952+ → (5+2) → 97-3*

97- → (9-7) → 23*

23* → 2*3 = $\boxed{6}$

2.3.2 Synthesized Attributes

We associate quantities with programming constructs (attributes)
↳ values
types.

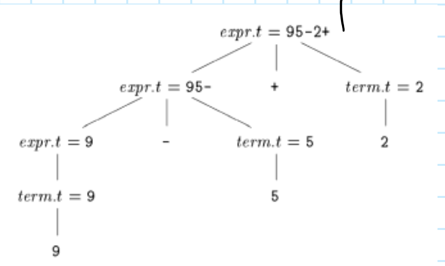
A syntax-directed definition associates:

1. With each grammar symbol, a set of attributes, and
2. With each production, a set of semantic rules for computing the values of the attributes on the symbols in the production.

Example: a node N is labeled by grammar symbol X.
↳ we write X.a to denote the value of attribute a of X
↳ a parse tree showing the attribute values at each node is called "annotated parse tree".

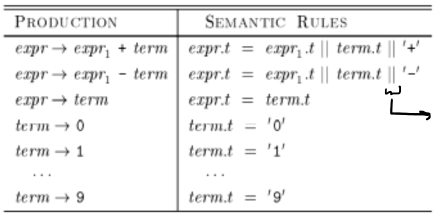
The following shows an annotated parse tree with an attribute "x" for nonterminals expr and term.

→ the value of the attribute at the root (x) is the postfix 95-2+



This annotated parse tree is based on the syntax directed definition below. for translating expressions of digits separated by +/- signs into postfix.

Each nonterminal has a string-valued attribute "t" that is the postfix notation.



PRODUCTION	SEMANTIC RULES
$expr \to expr_1 + term$	$expr.t = expr_1.t \,\|\|\, term.t \,\|\|\, '+'$
$expr \to expr_1 - term$	$expr.t = expr_1.t \,\|\|\, term.t \,\|\|\, '-'$
$expr \to term$	$expr.t = term.t$
$term \to 0$	$term.t = '0'$
$term \to 1$	$term.t = '1'$
...	...
$term \to 9$	$term.t = '9'$

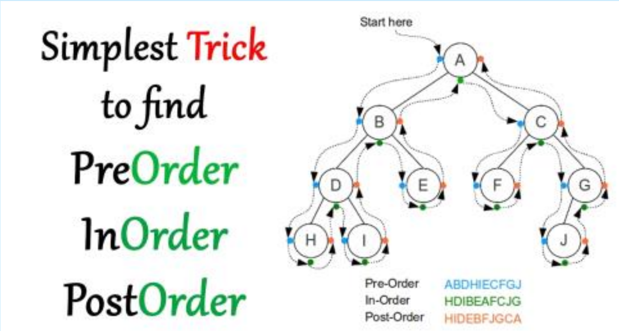
String concatenation

(Table: Syntax directed definition for infix to postfix)

[Basically, the table is a formal way of showing how to translate from infix to postfix notation.]

2.3.4 Tree Traversals

Tree traversal: Starts at the root and visits each node of the tree in a particular order.



"sdt"

→ A syntax directed translation scheme is a notation for specifying a translation by attaching program fragments to productions

↳ a translation scheme : is like a sdt scheme, except that the order of evaluation of the semantic rules is specified.

→ Program fragments in production bodies are "semantic actions"

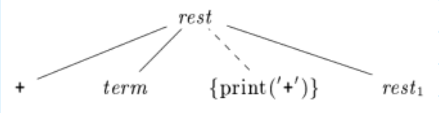
↳ the position at which the action needs execution is shown by enclosing it between {} and writing it into the production body:

$$rest \rightarrow + \; term \; \{ \; print('+') \; \} \; \underbrace{rest_1}_{\text{just to distinguish}}$$

→ When drawing a parse tree for a translation scheme;

- Indicate an action by putting an extra child for it, connected by a dashed line to the node of the head of the production.

its tree



Example:

tree : actions translating 9-5+2 to 95-2+

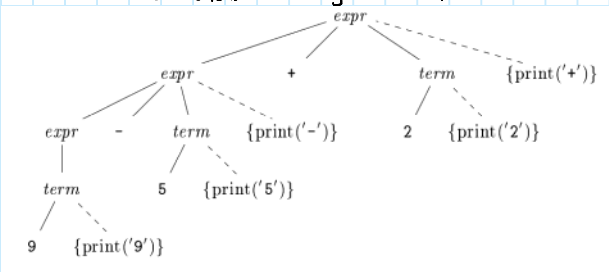
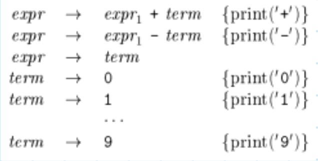


Table: actions for translating into postfix.



expr	→	*expr*₁ + *term*	{print('+')}
expr	→	*expr*₁ − *term*	{print('−')}
expr	→	*term*	
term	→	0	{print('0')}
term	→	1	{print('1')}
		...	
term	→	9	{print('9')}