CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

http://www.cse.buffalo.edu/faculty/alphonce/SP17/CSE443/index.php
https://piazza.com/class/iybn4ndqa1s3ei
Project adjustment

There's an inconsistency between the grammar and the example program. Did anyone spot it?
Project adjustment

- There's an inconsistency between the grammar and the example program. Did anyone spot it?

- Need to generate ';' at end of statement list:

  statement-list is:
  statement ';' statement-list | statement

  statement-list is:
  statement ';' statement-list | statement ';'
Project 2
due date adjustment

- Wednesday March 13 @ 5:00 PM

What hasn’t changed

- Homework 2 still due March 11 @ 5:00
- Project 3 still released March 11
Example 5.19 (p. 335)

$ \rightarrow \text{while ( C ) } S_1$

What are the semantics of this?
Example 5.19 (p. 335)

\[ S \rightarrow \text{while ( } C \text{ ) } S_1 \]

What are the semantics of this?

\[ L1 = \text{new()}; \]
\[ L2 = \text{new()}; \]
\[ S_1.\text{next} = L1; \]
\[ C.\text{false} = S.\text{next}; \]
\[ C.\text{true} = L2; \]
\[ S.\text{code} = \text{label || L1 || C.code || label || L2 || S}_1.\text{code} \]
Example 5.19 (p. 335)

\[ S \rightarrow \text{while ( } C \text{ ) } S_1 \]
$ S \rightarrow \text{while ( } C \text{ ) } S_1 $

The synthesized attribute $ S $.code is the [code] that [implements $S$]

The inherited attribute $C$.true labels the beginning of the code that must be executed if $C$ is true.

The inherited attribute $S$.next labels the beginning of the code that must be executed after $S$ is finished.

The synthesized attribute $S_1$.code is the [code] that [implements $S_1$] and ends with a jump to $S_1$.next

The inherited attribute $C$.false labels the beginning of the code that must be executed if $C$ is false.

The synthesized attribute $C$.code is the [code] that [implements $C$] and jumps either to $C$.true or to $C$.false, depending on whether $C$ is true or false.
"Syntax-directed translation schemes are a complementary notation to syntax-directed definitions. [...] A syntax-directed translation scheme (SDT) is a context-free grammar with program fragments embedded within production bodies." [p. 324]
Syntax-Directed Translation Schemes

"Any SDT can be implemented by first building a parse tree and then performing the actions in a [...] pre-order traversal." [p. 324]

"Typically, SDT's are implemented during parsing, without building a parse tree." [p. 324]
Syntax-Directed Translation Schemes

"...the simplest SDD implementation occurs when we can parse the grammar bottom-up and the SDD is S-attributed. In that case, we can construct an SDT in which each action is placed at the end of the production and is executed along with the reduction of the body to the head of that production." [p. 324]
Syntax-Directed Translation Schemes

"If the attributes are all synthesized, and the actions occur at the ends of the productions, then we can compute the attributes for the head when we reduce the body to the head." [p. 325]
Syntax-Directed Translation Schemes

"We consider [now] the more general case of an L-attributed SDD." [p. 331]

"The rules for turning an L-attributed SDD into an SDT are as follows:

1. Embed the action that computes the inherited attributes for a nonterminal A immediately before the occurrence of A in the body of the production.

2. Place the actions that compute a synthesized attribute for the head of a production at the end of the body of that production." [p. 331]
Implementing L-Attributed SDD's

"...we discuss the following methods for translating during parsing:

6. Implement an SDT in conjunction an LR parser. ... since the SDT for an L-attributed SDD typically has actions in the middle of productions, and we cannot be sure during an LR parse that we are even in that production until its entire body has been constructed ... [however] if the underlying grammar is LL, we can always handle both the parsing and translation bottom-up." [p. 338]
Bottom-up parsing of L-Attributed SDD's

"...given an L-attributed SDD on an LL grammar, we can adapt the grammar to compute the same SDD on the new grammar during an LR parse" [p. 348]

1. "Start with the SDT [...] which places embedded actions before each nonterminal to compute its inherited attributes and an action at the end of the production to compute synthesized attributes.

2. Introduce into the grammar a marker nonterminal in place of each embedded action. Each such place gets a distinct marker, and there is one production for any marker M, M → ε.

3. Modify the action a if marker nonterminal M replaces it in some production A → α {a} β, and associate with M → ε an action a' that

   (a) Copies, as inherited attributes of M, any attributes of A or symbols of α that action a needs.

   (b) Computes the attributes in the same way as a, but makes those attributes be synthesized attributes of M" [p. 349]
Bottom-up parsing of L-Attributed SDD’s

"...given an L-attributed SDD on an LL grammar, we can adapt the grammar to compute the same SDD on the new grammar during an LR parse” [p. 348]

1. "Start with the SDT [...] which places embedded actions before each nonterminal to compute its inherited attributes and an action at the end of the production to compute synthesized attributes.

2. Introduce into the grammar a marker nonterminal in place of each embedded action. Each such place gets a distinct marker, and there is one production for any marker M, M -> \( \varepsilon \).

3. Modify the action a if marker nonterminal M replaces it in some production A -> \( \alpha \{a\} \beta \), and associate with M -> \( \varepsilon \) an action a' that

   (a) Copies, as inherited attributes of M, any attributes of A or symbols of \( \alpha \) that action a needs.

   (b) Computes the attributes in the same way as a, but makes those attributes be synthesized attributes of M” [p. 349]
Bottom-up parsing of L-Attributed SDD's

"...given an L-attributed SDD on an LL grammar, we can adapt the grammar to compute the same SDD on the new grammar during an LR parse" [p. 348]

1. "Start with the SDT [...] which places embedded actions before each nonterminal to compute its inherited attributes and an action at the end of the production to compute synthesized attributes."

2. Introduce into the grammar a marker nonterminal in place of each embedded action. Each such place gets a distinct marker, and there is one production for any marker M, M -> ε.

3. Modify the action a if marker nonterminal M replaces it in some production A -> [a], and associate with M -> ε an action a' that

(a) Copies, as inherited attributes of M, any attributes of A or symbols of a that action a needs.

(b) Computes the attributes in the same way as a, but makes those attributes be synthesized attributes of M" [p. 349]
Example 5.25 [p. 349]

(put on board)
Example 5.26 [p. 349]

Needs figure 5.28 on page 336

Example 5.19 on page 335.