CS 434 Meeting 32— 5/1/02

Announcements

1. Midterm must be taken before midnight today.

Some Preliminaries to LR(1) parsing (cont.)

1. Preliminaries often involve definitions:

   nullable Given a grammar G, we say that a non-terminal N is *nullable* if $N \stackrel{*}{\Longrightarrow} \epsilon$.

   first set Given a grammar G and $\alpha \in (V_n \cup V_t)^*$ we define First(α) to be the set of terminals that might appear as the first symbol in a string derived from α. First(α) will include ϵ if $\alpha \stackrel{*}{\Longrightarrow} \epsilon$. Thus,

   $$\text{First}(\alpha) = \{a \in V_t \mid \alpha \stackrel{*}{\Longrightarrow} a\beta, \text{for some} \beta \in (V_n \cup V_t)^*\} \cup \{\epsilon \text{ if } \alpha \stackrel{*}{\Longrightarrow} \epsilon\}$$

2. Given that we have computed the set of nullable non-terminal, we can compute First for each terminal and non-terminal using a "run until nothing changes" algorithm: to compute First(α).

   - We use a table called 'FirstSet' with one entry for each terminal and non-terminal. Throughout the algorithm, for any $x \in (V_t \cup V_n)$
     $$\text{First}(x) \supseteq \text{FirstSet}[x]$$
   - FirstSet will be initialized as follows.
     (a) Set FirstSet[x] to $\{\epsilon\}$ for all nullable non-terminals and to $\{\}$ for all other non-terminals.
     (b) Set FirstSet[x] equal to $\{x\}$ for all terminals.
     (c) For each production of the form $N \rightarrow t\beta$ add t to FirstSet[N].
   - For each production of the form $N \rightarrow \beta$, write β as $\beta = \alpha\beta'$ where α is a string of nullable non-terminals, and β' is either the ϵ or a string of terminals and non-terminals beginning with a non-nullable symbol we will call M.
   - The approximations for the First sets stored in the FirstSet table are then improved until they become exact by repeating the following process until no further changes occur.
     – For each production $N \rightarrow \alpha\beta'$
       (a) For each $x \in \alpha$, add (FirstSet[x] $- \epsilon$) to FirstSet[N].
       (b) If $\beta' = \epsilon$ then add ϵ to FirstSet[N] otherwise add (FirstSet[M] $- \epsilon$) to FirstSet[N].

3. Consider how to construct these sets for the grammar:

   S → A B C
   A → a | CB
   B → C | A d | ϵ
   C → f | ϵ

   - All the non-terminals are nullable:
     – B and C are directly nullable.
     – The production $A \rightarrow BC$, then implies A is nullable.
     – Then, the production $S \rightarrow ABC$ implies S is nullable.
   - The productions break down as:

		α	β
S	→	A B C	
A	→		a
A	→	CB	
B	→	C	
B	→	A	d
B	→		ϵ
C	→		f
C	→		ϵ

- The FirstSet values start out as shown in the table below and can be refined by iterating over the production table.

S	A	B	C	a	d	f
{ ϵ }	{ ϵ, a }	{ ϵ }	{ ϵ, f }	{ a }	{ d }	{ f }
{ }	{ }	{ }	{ }	{ a }	{ d }	{ f }
{ }	{ }	{ }	{ }	{ a }	{ d }	{ f }
{ }	{ }	{ }	{ }	{ a }	{ d }	{ f }

4. The final preliminary is the definition of Follow:

$$\text{Follow}(N) = \{x \in V_t \mid A \stackrel{*}{\Longrightarrow} \alpha N x \beta\} \cup \{\epsilon \text{ if } S \stackrel{*}{\Longrightarrow} \alpha N\}$$

5. The computation of Follow(N) depends of the First sets defined earlier. If

$$M \rightarrow \alpha N \beta$$

then Follow(N) must contains First(β) . If β is nullable, then Follow(N) must also contain Follow(M). These observations are enough to give us an approximation algorithm for computing Follow. See the books on reserve for details.

SLR(1) Parsing

1. Given the notion of the "follow" set, we can illustrate the use of look-ahead in LR parsing, by considering the simplest form of look-ahead LR parsing — SLR(1) parsing (that's S for simple).

2. Recall the grammar

$$< \text{S} > \ \rightarrow \ \text{a} < \text{S} > \text{b} < \text{S} > \mid \epsilon$$

3. The LR(0) machine for this grammar contains three states with LR(0) conflicts (two of which were shown above):

   - The initial state is composed of the items:

     [< S' > → . < S > $]
     [< S > → . a < S > b < S >]
     [< S > → . ϵ]

   - Starting from the initial state on input "a" we reach the the state:

     [< S > → a . < S > b < S >]
     [< S > → . a < S > b < S >]
     [< S > → ϵ .]

   - Starting from the initial state on input "aSb" we reach the the state:

     [< S > → a < S > b . < S >]
     [< S > → . a < S > b < S >]
     [< S > → ϵ .]

4. For this grammar, a is not in Follow(S). So, we should never reduce using the production < S > → ϵ if the next "input" symbol is a. Therefore, the three states with LR(0) conflicts really don't have conflicts at all.

5. In general, we will say that a set of LR(0) items contains an SLR(1) conflict if either:

   (a) It contains two reduce items $[N \rightarrow \beta_1.]$ and $[M \rightarrow \beta_2.]$ such that the intersection of Follow(N) and Follow(M) is non-empty, or

   (b) It contains a reduce item $[N \rightarrow \beta_1.]$ and a shift item $[M \rightarrow \beta_2.x\beta_2]$ such that $x \in \text{Follow}(N)$.

6. If the LR(0) machine for a grammar G contains no states with SLR(1) conflicts we say that G is an SLR(1) grammar.

7. An SLR(1) parser for an SLR(1) grammar G behaves as follows in state π when the next input symbol is "x":

- reduce using production $N \rightarrow \beta$ if [$N \rightarrow \beta$.] $\in \pi$, and $x \in$ Follow(N).
- shift in next input if π contains one or more items of the form [$N \rightarrow \alpha . x \beta$].
- error otherwise.

LR(1) Parsing

1. Simply using Follow sets to interpret look ahead symbols may give less information than is really available.

   - Consider the grammar:

     E → (L , E)
     E → S
     L → L , E
     L → E
     S → ident
     S → (S)

     The state reached on input '(S' contains an SLR(1) conflict but 1 symbol look ahead is enough to allow us to parse.

     To see why, build the LR(0) machine. The conflict is between the items $[S \rightarrow (S.)]$ and $[E \rightarrow S.]$ and ")" is clearly in the Follow set of E. However, if one reduces using the production in the reduce item, one would quickly ends up in a state where you further reduce the E to an L. Then, you end up in a state where the only possible action is to shift in a comma. So, if the next input is not a comma, reducing by $E \rightarrow S$ is a dead end.

2. LR(1) parsing is a rather straightforward generalization of LR(0) parsing that keeps track of both what points in what productions we might be up to and what might follow the rhs's of the productions we are working on.

3. We start with plenty of new (but familiar) definitions.

   LR(1) item Given a grammar G, we say that $[N \rightarrow \beta_1 . \beta_2, a]$ is an *LR(1) item* or *LR(1) configuration* for G if $N \rightarrow \beta_1 \beta_2$ is a production in G and $a \in (V_t \cup \epsilon)$. The symbol 'a' is called the *lookahead*.

   Configuration Set We will refer to a set of LR(1) items as an LR(1) *configuration set*.

   Valid item Given a grammar G, we say that an LR(1) item $[N \rightarrow \beta_1 . \beta_2, a]$ is valid for $\gamma \in (V_n \cup V_t)^*$ if there is a rightmost derivation

   $$S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha N \omega \underset{\text{rm}}{\Longrightarrow} \alpha \beta_1 \beta_2 \omega$$

   such that $\alpha \beta_1 = \gamma$ and $a \in \text{First}(\omega)$.