CS 434 Meeting 31— 4/29/02

Announcements

1. Midterm must be taken before Wednesday.

The Correctness of LR(0) parsing

1. Lemma 2: For kernel items, $[N \rightarrow \beta_1.\beta_2] \in \Delta(\pi_0, \gamma)$ only if $[N \rightarrow \beta_1.\beta_2]$ is valid for γ.

2. Now, assuming Lemma 1, we can prove Lemma 2 by induction on the length of γ. The basis step is so simple that we will look at the induction step first:

   induction Assume that we know that the theorem holds for all strings of length n and consider some string γx such that γ is of length n and x is a single symbol.

   Suppose that $[N \rightarrow \beta_1 x.\beta_2]$ is an item in $\Delta(\pi_0, \gamma x)$. The fact that this item is in this set implies that the item $[N \rightarrow \beta_1.x\beta_2]$ must be in $\Delta(\pi_0, \gamma)$. This, together with our inductive assumption implies that $[N \rightarrow \beta_1.x\beta_2]$ must be valid for γ. Therefore, there exists a derivation:

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

   with $\alpha\beta_1 = \gamma$. This, however implies that $[N \rightarrow \beta_1 x.\beta_2]$ is indeed valid for γx.

   basis Similarly, when we consider strings of length 0, the only kernel item in $\Delta(\pi_0, \epsilon)$ is $[S' \rightarrow .S]$. The derivation $S' \underset{\text{rm}}{\overset{*}{\Longrightarrow}} S' \underset{\text{rm}}{\Longrightarrow} S$ shows that this item is valid for ϵ.

Some Preliminaries to LR(1) parsing

1. Consider the grammar

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

2. If we build the LR(0) machine for this grammar, we discover that it is not an LR(0) grammar because several states contain shift/reduce conflicts.

   - The initial state is composed of the items:

     $$[< \text{S'} > \ \rightarrow \ . < \text{S} > \$]$$
     $$[< \text{S} > \ \rightarrow \ . \ \text{a} < \text{S} > \text{b} < \text{S} >]$$
     $$[< \text{S} > \ \rightarrow \ . \ \epsilon]$$

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

     $$[< \text{S} > \ \rightarrow \ \text{a} \ . < \text{S} > \text{b} < \text{S} >]$$
     $$[< \text{S} > \ \rightarrow \ . \ \text{a} < \text{S} > \text{b} < \text{S} >]$$
     $$[< \text{S} > \ \rightarrow \ \epsilon \ .]$$

   We can use this machine anyway, if we are willing to look ahead a bit.

   - In all the sentential forms you can generate from the grammar an "a" will never directly follows and S.
   - As a result, in either of the states shown, choosing to reduce when the next input is an "a" would definitely lead to a dead end.

3. In general, suppose that we find that after reading some prefix ω_1 of an input $\omega_1 x \omega_2$ we end up in a state that contains a reduce item $[N \rightarrow \beta.]$ which conflicts with some other item.

   - If we decide to reduce using the production in this item, we are basically assuming that

     $$S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha N x \omega_2 \underset{\text{rm}}{\Longrightarrow} \alpha\beta x \omega_2 \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \omega_1 x \omega_2$$

   - That is, we are assuming that there is some sentential form in which an x can follow an N.

4. 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,

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

5. The set of nullable non-terminals can be computed by the following algorithm:

   (a) Set "nullable" equal to the set of non-terminals appearing on the left side of productions of the form $N \rightarrow \epsilon$.

   (b) Until doing so adds no new non-terminals to "nullable", examine each production in the grammar adding to "nullable" all left-hand-sides of productions whose right-hand-side consist entirely of symbols in "nullable".

6. 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].