CS 434 Meeting 33— 5/3/02

Announcements

1. Colloquium speaker from GE (visualization research)

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 $a \in \text{First}(\omega)$.

Building an LR(1) Machine

1. First, we need to extend the definitions which we used to define the transition function for an LR(0) machine to account for the lookaheads we have added to LR(1) items.

   goto Given a set of LR(1) items for a grammar G, we define

   $$goto(\pi, x) = \{[N \rightarrow \beta_1 x.\beta_2, a] \mid [N \rightarrow \beta_1.x\beta_2, a] \in \pi\}$$

   closure Given a set π of LR(1) items for a grammar G with productions P, we define closure(π) to be the smallest set of LR(1) items such that:

   (a) closure(π) $\supseteq \pi$
   (b) if $[N_1 \rightarrow \beta_1.N_2\beta_2, a] \in$ closure(π) and $N_2 \rightarrow \beta_3 \in P$ then, for each $b \in \text{First}(\beta_2 a)$, $[N_2 \rightarrow .\ \beta_3, b] \in$ closure(π)

2. With these definitions, it should be obvious, that the next step is to define the LR(1) finite automaton for a grammar G consisting of:

   - A set of states with one state for every subset of LR(1) items.

- An alphabet consisting of the terminals and non-terminals of G.
- A set of final states consisting of the set of all states except the state corresponding to the empty set of LR(1) items.
- A transition function defined by:

$$\delta(\pi, x) = closure(goto(\pi, x))$$

- The state closure($[S' \rightarrow .S\$, \epsilon]$) as its initial state.

3. The notions of a kernel item and a reduce item transfer naturally from LR(0) items to LR(1) items.

4. Note that the language accepted by the LR(1) FSM is the same as that accepted by the LR(0) machine (i.e. the set of viable prefixes). The extra states in the machine, however, include information that can be used to make a better parser.

5. Consider what happens when we build the LR(1) machine for the non-SLR(1) grammar considered yesterday.

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

6. A set of LR(1) items contains a conflict if it contains a reduce items of the form $[N \rightarrow \beta_1., x]$ and either another reduce item of the form $[M \rightarrow \beta_2., x]$ or a shift item of the form $[M \rightarrow \alpha.x\beta_2, y]$.

7. We say that a grammar is LR(1) if the reachable states in its LR(1) machine are conflict free.

8. Given an LR(1) grammar, its LR(1) parser, shifts in state π with input x if state π contains a shift item of the form $[N \rightarrow \alpha.x\beta, a]$, reduces using production $N \rightarrow \beta$ if state π contains a reduce item of the form $[N \rightarrow \beta., x]$ and reports error otherwise.

A Word About LALR(1) Parsing

1. We've got a problem...
   - The machines that result from the LR(1) construction tend to be too large to use in practice.
   - The SLR(1) technique is too weak (i.e. there are too many grammars that could be parsed deterministically using LR(1) techniques whose LR(0) machines still have SLR(1) conflicts).

   LALR parsing attempts to find a happy medium.

2. Given a set of LR(1) items, we can obtain a set of LR(0) items by dropping the lookahead from each LR(1) item. We call this set the *core* of the original set of LR(1) items.
   - As an example, the core of the state in the LR(1) machine reached upon reading "(S" is just the state of the LR(0) machine reached on the same input.

3. The core of each state in the LR(1) machine will correspond to some state in the LR(0) machine.

4. Many states in the LR(1) machine may have the same state of the LR(0) machine as their core.

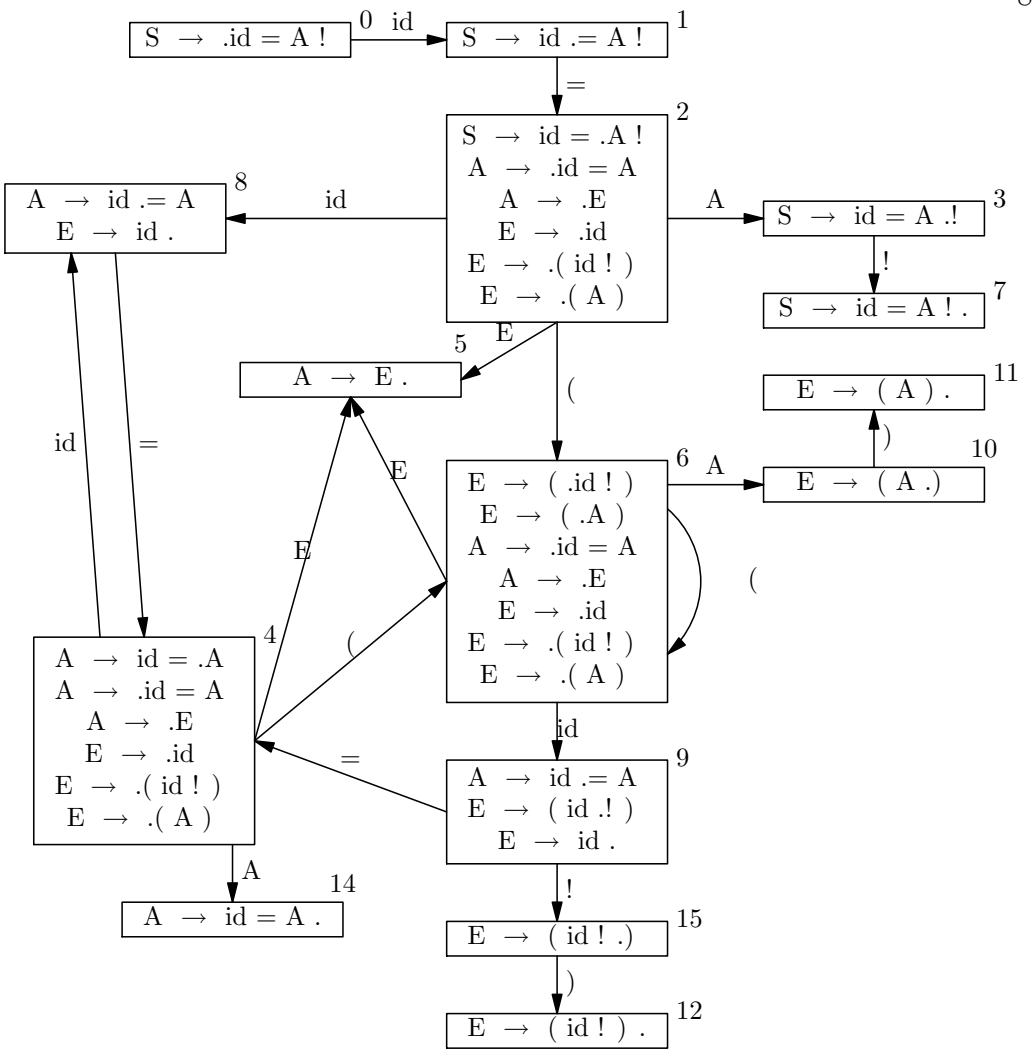
5. Using these facts, for each state in the original LR(0) machine for a grammar we can define a corresponding set of LR(1) items. For an LR(0) state π, we will use the set of LR(1) items defined by

$\{[N \rightarrow \beta_1.\beta_2, x] |$ for some π' in the LR(1) machine, $[N \rightarrow \beta_1.\beta_2, x] \in \pi'$ & $\pi = core(\pi')\}$

6. The LALR(1) machine for a grammar is formed by replacing each of the sets of LR(0) items associated with the states of the LR(0) machine with sets of LR(1) items in the way just described.

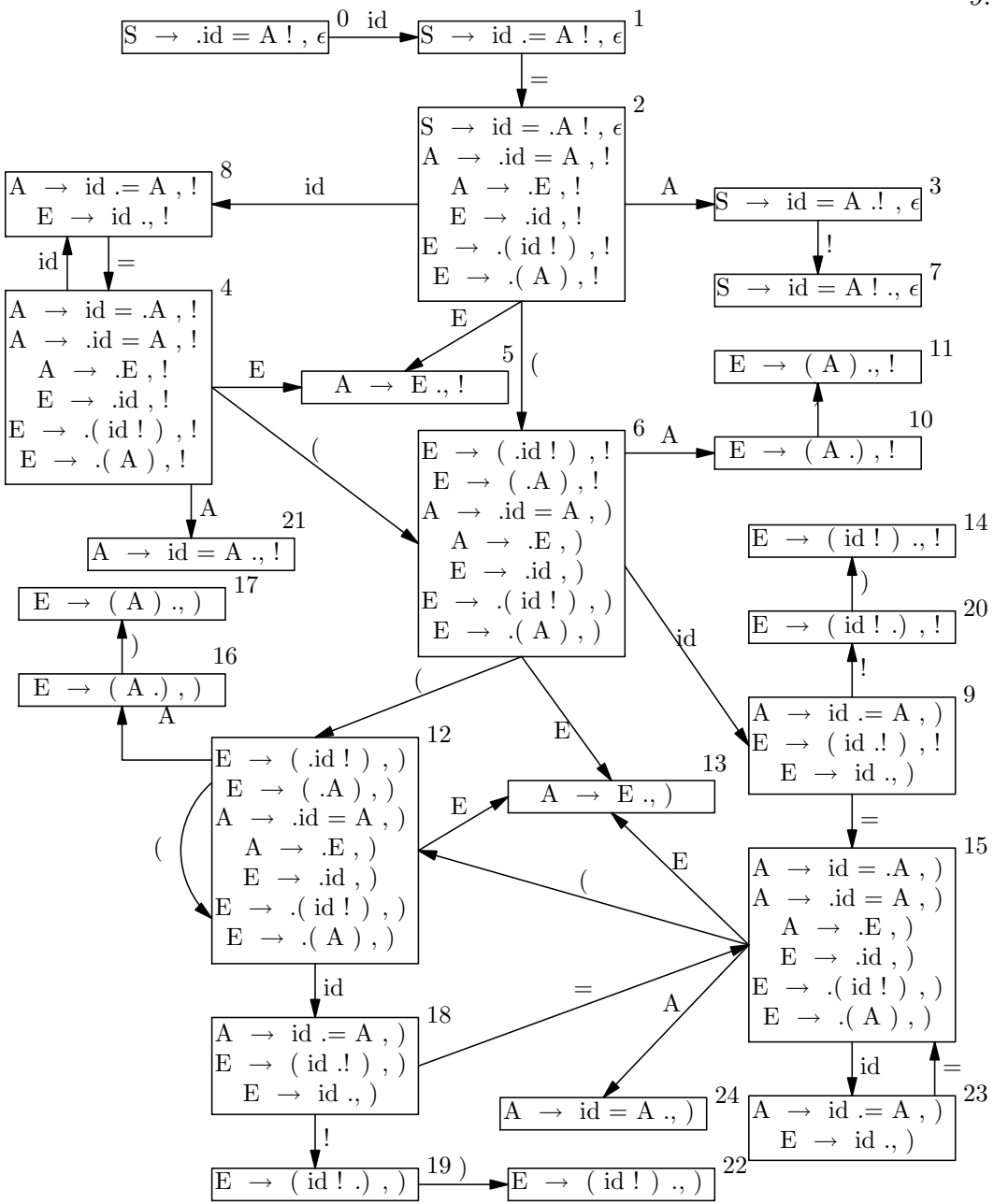
7. Consider how this works on the grammar:

S → id = A !
A → id = A
  | E
E → id
  | (id !)
  | (A)



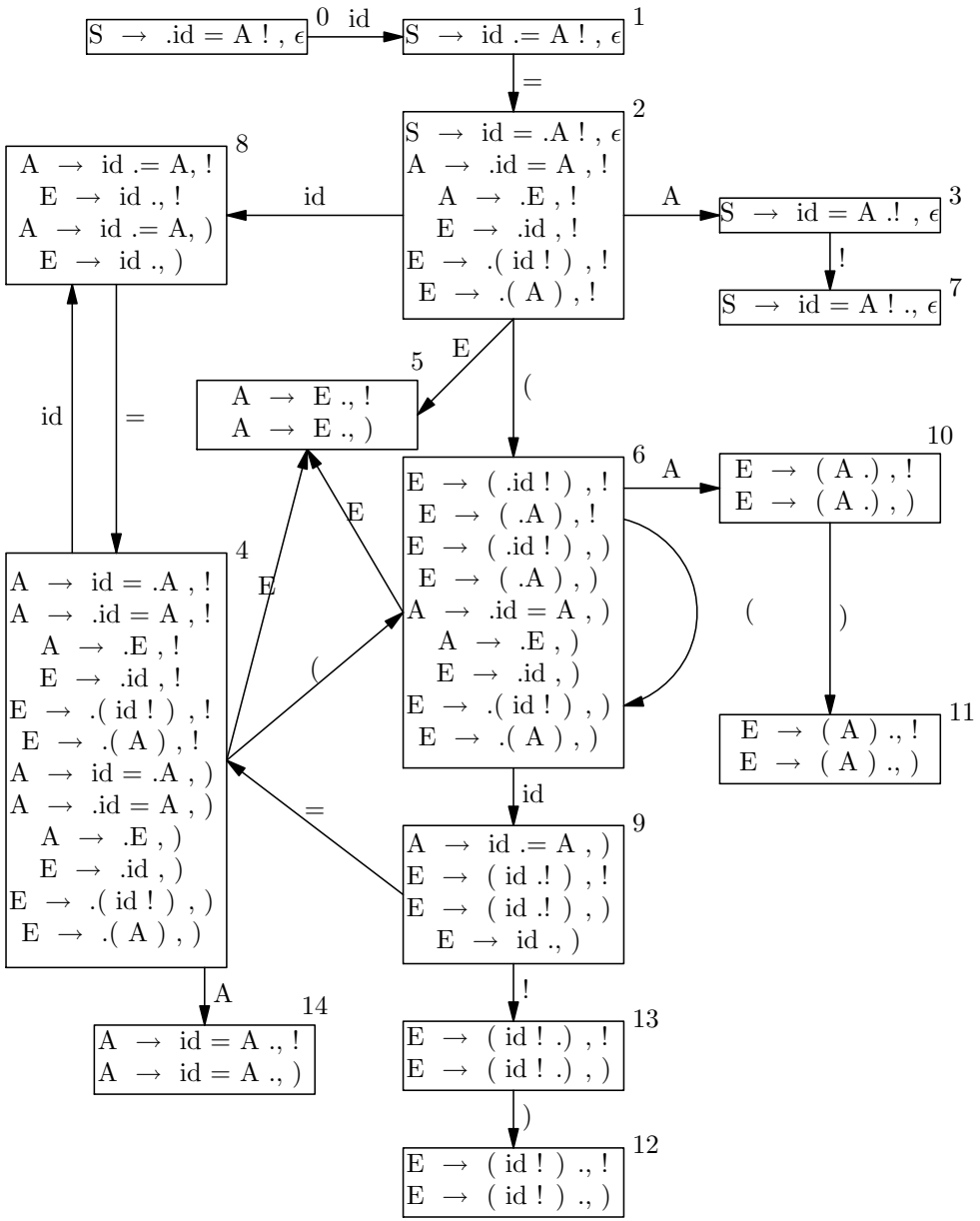
8. The mess to the left is the LR(0) machine for the grammar. Note that two states (8 and 9) contain LR(0) conflicts and that one of them (9) is also an SLR(1) conflict since and E can be followed by either an ")" or an exclamation point.

.



9. The even bigger mess on this page is (hopefully) the LR(1) machine for the grammar. While it is ugly, it contains no conflicts. The following handy guide lists the states of the LR(0) machine and the states of the LR(1) machine to which they correspond.

0 0
1 1
2 2
3 3
4 4, 15
5 5, 13
6 6, 12
7 7
8 8, 23
9 9, 18
10 10, 16
11 11, 17
12 14, 22
13 20, 21
14 24, 25



10. Finally, here we see the LALR(1) machine for this grammar. The states are numbered to match the numbering of the states from the LR(0) machine. Each state contains the union of the LR(1) items found in the LR(1) states whose core is equivalent to the corresponding LR(0) state.

    Note, there are no LALR(1) conflicts.