Conceptual Issues

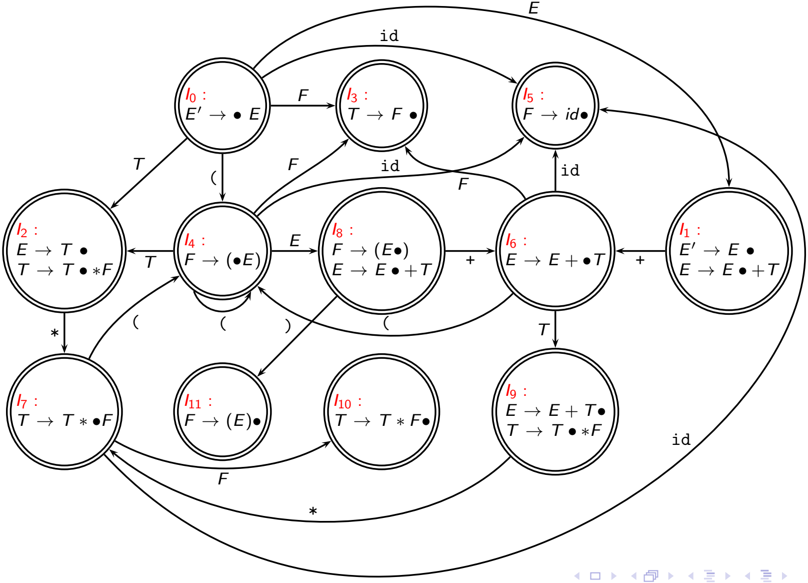
1. What information do the states contain?
2. Where exactly is handle detection taking place in the parser?
3. Why is *FOLLOW* information used to create the reduce entries in the action table ?

To answer these questions, we need to see the canonical collection of LR(0) items as a DFA.

- A node labeled I_i is constructed for each member of C.
- For every nonempty goto$(I_i, X) = I_j$, a directed edge (I_i, I_j) is added labeled with X.
- The graph is a deterministic finite automaton if the node labeled I_0 is treated as the *start* state and all other nodes are made final states.

What does the automaton recognize?

The DFA of an LR parser



Viable Prefix and Valid Items

A *viable prefix* is the prefix of a right-sentential form that does not contain any symbols to the right of a handle.

Viable Prefix and Valid Items

A *viable prefix* is the prefix of a right-sentential form that does not contain any symbols to the right of a handle.

The automation shown before recognizes viable prefixes only.

Viable Prefix and Valid Items

A *viable prefix* is the prefix of a right-sentential form that does not contain any symbols to the right of a handle.

The automation shown before recognizes viable prefixes only.

1. By adding terminal symbols to viable prefixes, rightmost sentential forms can be constructed.

Viable Prefix and Valid Items

A *viable prefix* is the prefix of a right-sentential form that does not contain any symbols to the right of a handle.

The automation shown before recognizes viable prefixes only.

1. By adding terminal symbols to viable prefixes, rightmost sentential forms can be constructed.
2. Viable prefixes are precisely the set of symbols that can ever appear on the stack of a LR parser

Viable Prefix and Valid Items

A *viable prefix* is the prefix of a right-sentential form that does not contain any symbols to the right of a handle.

The automation shown before recognizes viable prefixes only.

1. By adding terminal symbols to viable prefixes, rightmost sentential forms can be constructed.
2. Viable prefixes are precisely the set of symbols that can ever appear on the stack of a LR parser
3. A viable prefix either contains a handle or contains a part of a handle.

Viable Prefix and Valid Items

A *viable prefix* is the prefix of a right-sentential form that does not contain any symbols to the right of a handle.

The automation shown before recognizes viable prefixes only.

1. By adding terminal symbols to viable prefixes, rightmost sentential forms can be constructed.
2. Viable prefixes are precisely the set of symbols that can ever appear on the stack of a LR parser
3. A viable prefix either contains a handle or contains a part of a handle.
4. For a viable prefix, if is useful to identify the portion of the handle that it contains.

Viable Prefix and Valid Items

A LR(0) item $A \rightarrow \beta_1 \bullet \beta_2$ is defined to be *valid* for a viable prefix, $\alpha\beta_1$, provided $S \overset{*}{\Rightarrow}_{rm} \alpha A\ \mathtt{w} \Rightarrow_{rm} \alpha\beta_1\beta_2\ \mathtt{w}$

1. There could be several distinct items which are valid for the same viable prefix γ.
2. It is interesting to note that in above, if $\beta_2 = B\gamma$ and $B \rightarrow \delta$, then $B \rightarrow \bullet\delta$ is also a valid item for this viable prefix.
3. A particular item may be valid for many distinct viable prefixes.

This statement means that if we have a partially recognized production A -> beta1 • beta2, then whenever this item is valid for a viable prefix, the first production of B (i.e., B -> • delta) must also be included in the valid items for that viable prefix, because after recognizing beta1, the next expected symbol is B, which must expand according to its production rule.

Viable Prefixes and Valid Items

- For the LR-automaton shown earlier, consider the path labeled by the viable prefix $(E+$ ending in I_6. The items valid for $(E+$ are:
  1. $E' \Rightarrow_{rm} E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow (E+T)$ shows that $E \rightarrow E + \bullet T$ is a valid item for $(E+$.
  2. $E' \Rightarrow E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow (E+T) \Rightarrow (E+T*F)$ shows that $T \rightarrow \bullet T * F$ is also a valid item.
  3. $E' \overset{*}{\Rightarrow}_{rm} (E+T) \Rightarrow (E+F)$ shows that $T \rightarrow \bullet F$ is another such item.
  4. $E' \overset{*}{\Rightarrow}_{rm} (E+T) \Rightarrow (E+F) \Rightarrow (E+(E))$ shows that $F \rightarrow \bullet (E)$ is also a valid item for $(E+$.
  5. Finally, $E' \overset{*}{\Rightarrow}_{rm} (E+F) \Rightarrow (E+\texttt{id}\ )$ shows that $F \rightarrow \bullet\ \texttt{id}$ is a valid item for $(E+$.

  It should be noted that are no other valid items for this viable prefix.

Viable Prefixes and Valid Items

In an LR parser, a complete item means that the dot (•) is at the end of a production, indicating that a production is fully recognized and should be reduced. However, not all lookahead symbols necessarily trigger a reduction. The parser must determine when to apply the reduction based on which symbols can legally follow the reduced non-terminal in a valid derivation.

Given a LR(0) item, say $T \rightarrow T \bullet * F$, there may be several viable prefixes for which it is valid.

1. $E' \Rightarrow_{rm} E \Rightarrow T \Rightarrow T * F$ shows that this item is valid for the viable prefix T .
2. $E' \Rightarrow E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow (T) \Rightarrow (T * F)$ shows that it is also valid for (T .
3. $E' \Rightarrow E \Rightarrow T \Rightarrow T * F \Rightarrow T * (E) \Rightarrow T * (T) \Rightarrow T * (T * F)$ shows that it is valid also for T * (T .
4. $E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T * F$ shows validity for $E + T$.

There may be several other viable prefixes for which this item is valid.

If we wrongly apply a reduction without considering lookahead, we might make incorrect reductions, leading to incorrect parsing decisions. This is why LR(1) parsing includes lookahead symbols in states to determine when reductions should be applied.

Theory of LR Parsing

THEOREM : Starting from I_0, if traversing the LR(0) automaton γ results in state j, then set items in I_j are the only valid items for the viable prefix γ.

- The theorem stated without proof above is a key result in LR Parsing. It provides the basis for the correctness of the construction process we learnt earlier.
- An LR parser does not scan the entire stack to determine when and which handle appears on top of stack (compare with shift-reduce parser).
- The state symbol on top of stack provides all the information that is present in the stack.
- In a state which contains a complete item a reduction is called for. However, the lookahead symbols for which the reduction should be applied is not obvious.
- *In SLR(1) parser the FOLLOW information is used to guide reductions.*