CSE431 – Translation of Computer Languages

- SLR(1)
- LALR(1)
- LR(1)

Doug Shook

Conflict review

- What are the two types of conflicts that can arise when creating a parse table?
- What are the potential causes of these conflicts?

SLR(k)

- Simple LR with k tokens of lookahead
  - k is typically 1
- Process:
  - Generate LR(0) as normal
  - For inadequate states:
    - Verify that the inadequacy is not due to ambiguity
    - Examine the reductions
      - We should take the reduction if the lookahead symbol is in the FOLLOW set

Example:

- Let's construct the CFSM for the following grammar:

```
Start  -> E $
E      -> E plus T
       |   T
T      -> T times num
       |   num
```

- Identify the states with conflicts
  - Convince yourself that they are not due to ambiguity
  - Construct the FOLLOW sets as needed
  - Create the table
  - Try parsing "num plus num times num $"

LALR(k)

- SLR is still not powerful enough for all grammars
- Craft the CFSM for the following grammar:

```
Start -> S $
S     -> AB
       |   a c
       |   x A c
A     -> a
B     -> b
       | lambda
```

- What happens when you try to use SLR?

LALR(k)

- Still based on LR(0)
  - Will have as many states as LR(0)
- Includes two new ideas
  - ItemFollow
  - Lookahead propagation graph
- Main idea: instead of using FOLLOW from the grammar, compute ItemFollow using CFSM states

ItemFollow

- Each vertex in the graph contains information about the state (from CFSM) and the item
  - Represented as 2-tuple <state, item>

- ItemFollow(v)
  - What symbols could follow this item after a reduction from this state?

ItemFollow

- What will ItemFollow be for items that arrive via closure?
  - How about the kernel?
  - What about lambda?
- The propagation graph is generated during the same pass that ItemFollow is initialized
  - Propagation comes afterwards

Pseudocode

- Create CFSM
  - Each <state, item> pair becomes a vertex
  - Initialize ItemFollow(<0, Start>) = {$}

```
For each state s
   For each item of form A→α●Bβ
      vertex u=<s, A→α●Bβ>
      vertex v=<s', A→αB●β>, successor of u on
      shift(B)
      Add edge (u,v)
      For each vertex w of form <s,B→●δ>
         Add FIRST(β) to ItemFollow(w)
         if β derives λ
            add edge (u,w)
```

(pseudocode provided by Michael Wilson)

Propagation

- Once the last step is complete, propagate information along graph edges
  - Propagate symbols down edges until no further changes are made
- Once this is complete, we can construct the parse table
  - Shift is handled normally
  - Each reduction should only be taken if lookahead is in ItemFollow

LR(k)

- LALR(k) is not the most powerful form of parse table creation
  - LR(k) is

- We've seen LR(0)
  - LR(k) (k > 0) uses the same process but with lookahead
  - In theory, can handle more grammars than LALR
  - In practice, it is not often used.
    - Aren't many grammars that are LR(k) but not LALR
    - LR(k) has a lot more states, more space

Practice Problems

- Construct the SLR(1) parse table for the following:

```
S -> A$
A -> aBc
   | abb
B -> b
```

Practice Problems

- Is the following grammar SLR(1)? Is it LALR(1)?
  - Construct the parse table

```
P -> S$
S -> L = R
     | R
L -> id
     | * R
R -> L
```