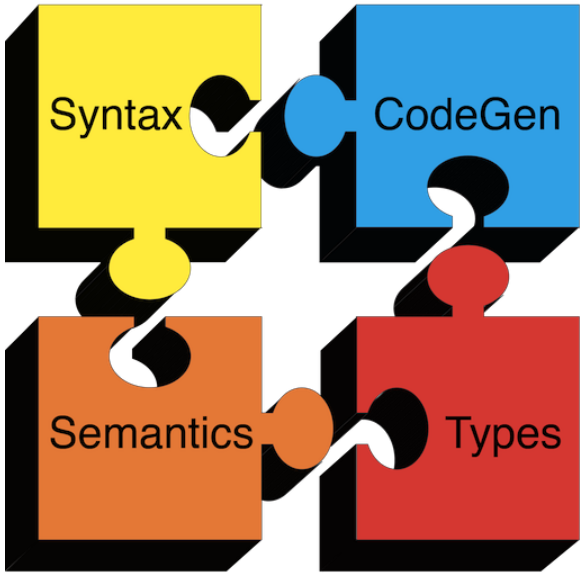

Compilers

SLR Improvements

SLR Improvements

- Rerunning the viable prefixes automaton on the stack at each step is wasteful
  - Most of the work is repeated

SLR Improvements

- Remember the state of the automaton on each prefix of the stack

- Change stack to contain pairs

$$\langle \text{Symbol, DFA State} \rangle$$

SLR Improvements

- For a stack

$$\langle \text{sym}_1, \text{state}_1 \rangle \ldots \langle \text{sym}_n, \text{state}_n \rangle$$

state_n is the final state of the DFA on $\text{sym}_1 \ldots \text{sym}_n$

- Detail: The bottom of the stack is ⟨any,start⟩ where
  - any is any dummy symbol
  - start is the start state of the DFA

SLR Improvements

- Define goto[i,A] = j if $state_i \rightarrow^A state_j$

- goto is just the transition function of the DFA
  - One of two parsing tables

SLR Improvements

- Shift x
  - Push $\langle a, x \rangle$ on the stack
  - a is current input
  - x is a DFA state
- Reduce $X \rightarrow \alpha$
  - As before
- Accept
- Error

SLR Improvements

For each state s_i and terminal a

- If s_i has item $X \rightarrow \alpha.a\beta$ and goto[i,a] = j then action[i,a] = shift j
- If s_i has item $X \rightarrow \alpha.$ and $a \in$ Follow(X) and $X \neq S'$ then action[i,a] = reduce $X \rightarrow \alpha$
- If s_i has item $S' \rightarrow S.$ then action[i,$] = accept
- Otherwise, action[i,a] = error

SLR Improvements

```
Let I = w$ be initial input
Let j = 0
Let DFA state 1 have item S' → .S
Let stack = ⟨ dummy, 1 ⟩
    repeat
        case action[top_state(stack),I[j]] of
            shift k:  push ⟨ I[j++], k ⟩
            reduce X → A:
                pop |A| pairs,
                push ⟨X, goto[top_state(stack),X]⟩
            accept: halt normally
            error: halt and report error
```

SLR Improvements

- Note that the algorithm uses only the DFA states and the input
  - The stack symbols are never used!

- However, we still need the symbols for semantic actions

SLR Improvements

- Some common constructs are not SLR(1)

- LR(1) is more powerful
  - Build lookahead into the items
  - An LR(1) item is a pair: LR(0) item x lookahead
  - [T→ . int * T, $] means
    - After seeing T→ int * T reduce if lookahead is $
  - More accurate than just using follow sets
  - Take a look at the LR(1) automaton for your parser!