Project, Phase 1 Clarifications

CS 3240

Spring 2013

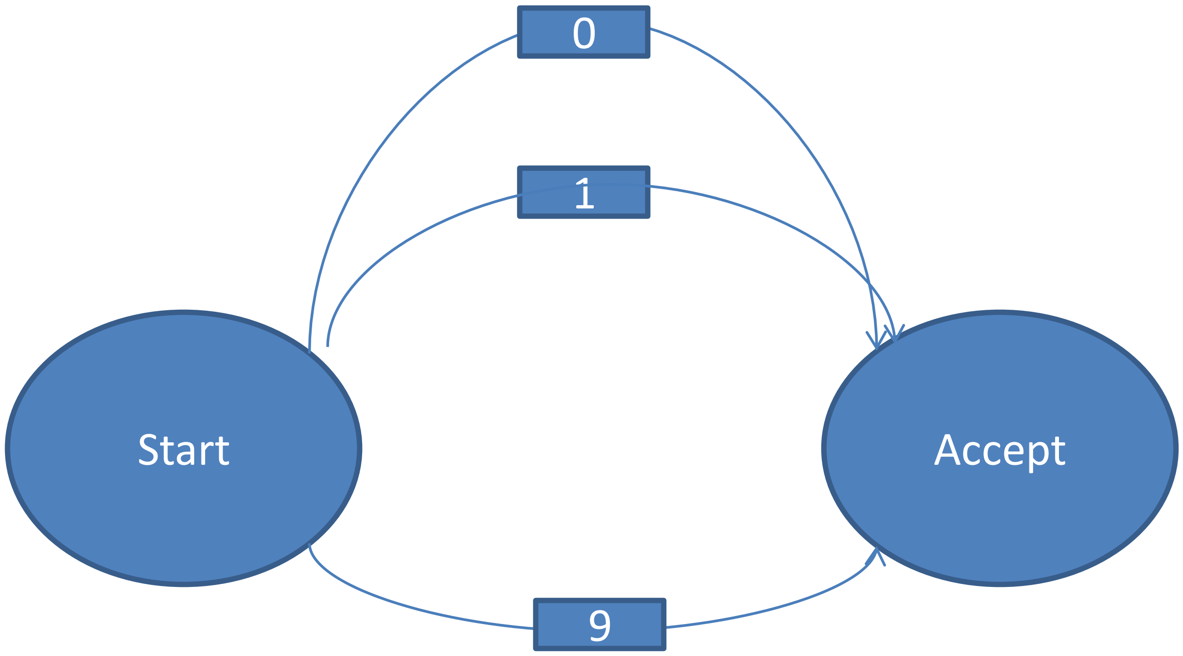
Given: 3 components

- Grammar in the appendix of the project write-up
  - Tells you the parsing rules of the regular expressions down to the character classes
- Spec (sample uploaded in T-square)
  - Character classes and reg exps used to build the scanner (giant NFA) using a scanner generator (recursive descent parser)
- Input (sample uploaded in T-square)
  - Identify each token or character class

Step 1 - Primitive NFAs

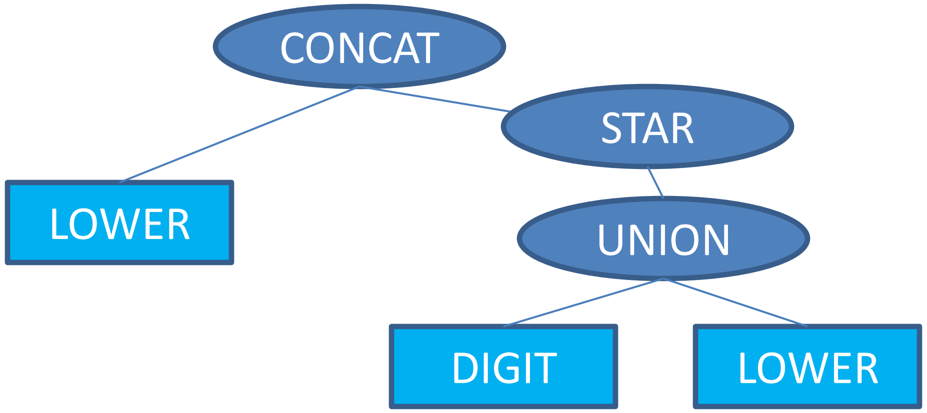
- Read the spec and build primitive NFAs for each of the character classes
  - Say, DIGIT would be represented by a NFA that has a start and an accept state
  - Any character 0-9 causes a transition from the start to the accept state

Primitive NFA - Example



Step 2 - Parsing the Reg Exps (1)

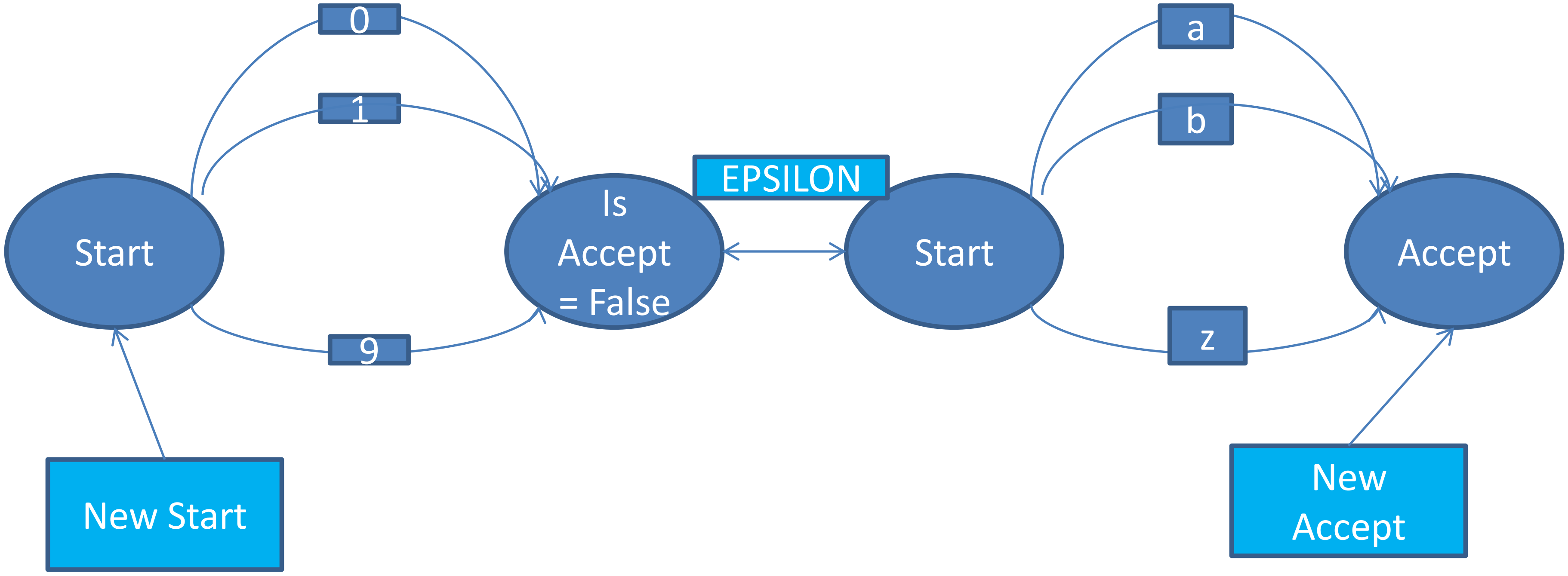
- Use the rules specified in the grammar and a recursive descent parser to build the scanner for each regular expression in the spec
  - Say, the token is $IDENTIFIER $LOWER ($DIGIT | $LOWER)*
  - The grammar + recursive descent parser gives the parse tree for it



Step 2 - Parsing the Reg Exps (2)

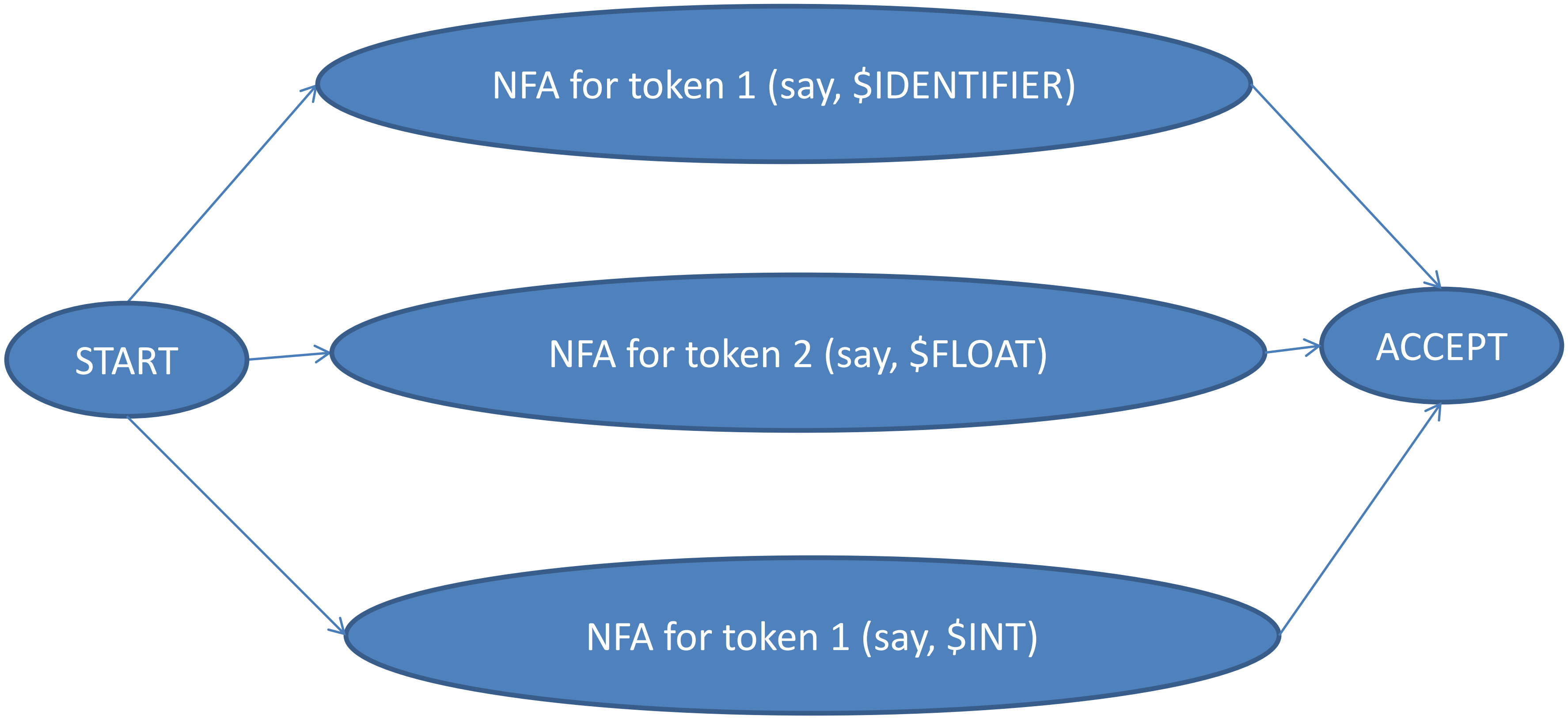
- Use this parse tree to join your primitive NFAs
  - You could do this in a single step while you are doing the recursive descent parsing
  - Or, build the parse tree and then walk the tree to join the primitive NFAs

Joining primitive NFAs for a reg exp – Example (DIGIT|LOWER)



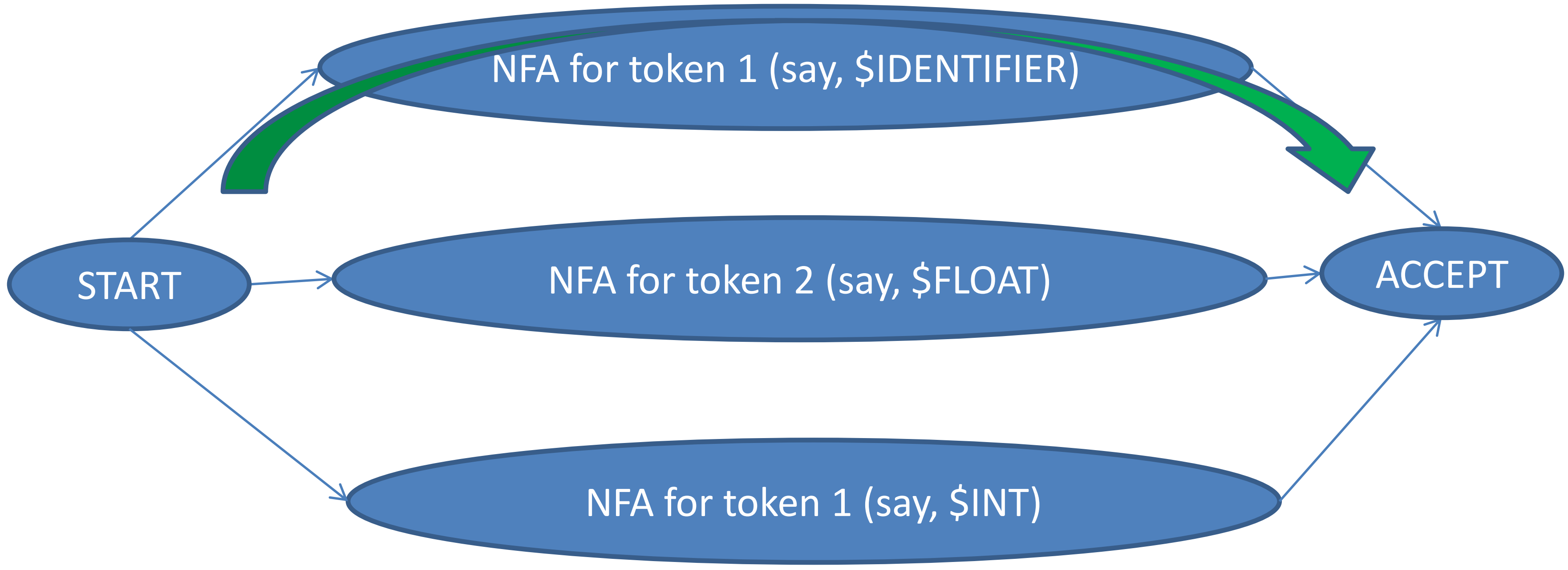
Step 3 - Giant NFA

- Union of NFAs of all reg exps


START
NFA for token 1 (say, $IDENTIFIER)
NFA for token 2 (say, $FLOAT)
NFA for token 1 (say, $INT)
ACCEPT

Step 4 - Input Test

- Say, a99z
  - Is a valid identifier
  - Should reach the accept state via the IDENTIFIER path in the giant NFA, and print the token name (that is IDENTIFIER)


NFA for token 1 (say, $IDENTIFIER)
START
NFA for token 2 (say, $FLOAT)
ACCEPT
NFA for token 1 (say, $INT)