CS 415: Lecture 3

- Lexical Analysis, part 1

Project

- The overall description of the project and phase 1 has been posted to the web
  - Print project1.pdf
  - Download project1.tar.gz from the web
- Need to recreate your account using
  [http://remus.rutgers.edu/newaccount.html](http://remus.rutgers.edu/newaccount.html)
  - Your accounts from last semester will disappear after the first week of February
- Phase 1 is due at 7pm on Wed 2/9
  - Instructions for handing in will be on web shortly
  - NO LATE ASSIGNMENT WILL BE ACCEPTED!!
  - Should probably hand in a bit early this time around to make sure that you can get "handin" to work correctly for you
Lexical Analyzer

- Typically implemented as a coroutine of the parser
- Separate input character stream into tokens
  - Tokens may have attributes. Scanner also pass attributes to parser.
  - Discard comments and "token separators" (typically white space)

Terminology

- **token**: terminal symbol in CFG of language
- **lexeme**: character sequence in input defining a token
- **pattern**: rule describing the set of possible lexemes that match a given token
Regular Expressions

- Patterns are typically defined by regular expressions
- Recall that regular expressions represent languages
- Some more terminology:
  - **Alphabet**: finite set of symbols
  - **String**: finite sequence of symbols from an alphabet
  - **Language**: set of strings over an alphabet
- Operations include union, concatenation, and Kleene closure
  - Do you remember the definitions of these operations?
  - Order of precedence: closure, concatenation, union

Rules for Defining REs

- Assume that we are talking about REs defined for an alphabet A
- If r is an RE, then we say that L(r) is the language defined by r (that is, all strings that matches the pattern defined by r)
- ε is an RE denoting the set {ε}
- If a ∈ A, then a is an RE denoting {a*}
- If r and s are REs, denoting L(r) and L(s), then:
  - (r) is an RE denoting L(r)
  - rs is an RE denoting L(r)∪ L(s)
  - rs is an RE denoting L(r)L(s)
  - r* is an RE denoting L(r)*
- A language defined by a regular expression is called a regular set
Recognizers

- A recognizer for a language is a program that takes as input a string \( x \) and outputs "yes" if \( x \) is a sentence of the language and "no" otherwise.
- REs can be recognized by finite automata.
- Remember what an FA is?
- What's the FA for recognizing identifiers defined as follows?
  1. letter -> \( \{a \mid b \mid c \mid \ldots \mid z \mid A \mid B \mid C \mid \ldots \mid Z\} \)
  2. digit -> \( \{0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\} \)
  3. id -> letter ( letter | digit )

Finite Automata

- A finite automaton is a mathematical model that consists of:
  1. \{states\}
  2. \{input symbols\}
  3. \{transitions\}
  4. start state
  5. \{final states\}
- Transitions between states occur on specific input symbols.
FA and Languages

- The language recognized by an automaton is the set of strings it accepts by starting in the start state, using transitions corresponding to input symbols in the input string, and processing all input and finishing in a final state.

RE for integers: [ 0-9 ]+

A Little Exercise

- Let's construct an FA to recognize the regular expression (a|b)*abb
**Nondeterministic vs Deterministic**

- What kind of FA did we just construct?
- Two kinds of automata: **deterministic** and **nondeterministic**
  - Deterministic automata have only 1 transition per state on a specific input and do not allow transition on the empty string
  - What about nondeterministic automata?
- A DFA is a special case of an NFA
  - No ε transitions
  - Single-valued transition function
- How can we build a program to “run” NFAs?
  - A (surprising?) result: can simulate an NFA using a DFA!
  - We’ll look at RE → NFA and then NFA → DFA

**RE to NFA Conversion**

- Straightforward translation using composition operators of REs
- Let N(r) be the NFA that recognizes the language L(r) or an RE r
  - Assume that N(r) has only a single final state
  - Suppose some NFA has multiple final states, how can we change it to an NFA that has only a single final state?
For RE $\varepsilon$, $s \xrightarrow{\varepsilon} f$

For RE $a$, terminal symbol, $s \xrightarrow{a} f$

For $w,t$ REs with corresponding NFAs $N(w), N(t)$, $w | t$ yields, $s' \xrightarrow{\varepsilon} f'$

For $w$ RE, $w^*$ yields, $s' \xrightarrow{\varepsilon} f'$

For $(w)$ RE, use $N(w)$. 

Rutgers University, DCS CS 415: Compilers
NFA to DFA Conversion

- Deterministic computation is desirable if we want a write a scanner as a program
  - Need to convert NFA to equivalent DFA
  - Then can simulate DFA recognition process using tables in program to describe transitions
  - If process ends up in a final state, a token has been recognized

---

NFA to DFA Conversion

- How does an NFA compute?
  - Start off in the start state
  - Compute set S of all states reachable on \( e \) transitions.
  - Given next input symbol is \( a \), calculate set of states \( T \), reachable as transition \( (s,a) \) where \( s \in S \)
  - Repeat steps 2,3 until input is exhausted. If final set of states contains a final state, then string has been recognized.

- Intuition: simulate an NFA on a DFA by having a state per set of possible states
NFA to DFA Conversion

- Whenever there is an \( \varepsilon \) transition out of a state \( s \), the NFA may go to any of the states reachable in this manner without consuming any input symbols. Call these states the \( \varepsilon \)-closure of state \( s \).
  - By looking at \( \varepsilon \)-closures, we form sets of related states in the NFA; these become states in the corresponding DFA.
  - Edges in the DFA correspond to sets of edges in the NFA (connecting different \( \varepsilon \)-closure sets of states).

Conversion Algorithm

- Need two primitive functions
  - \( \varepsilon \)-closure(\( T \)), for \( T \) a set of states in the NFA
    - Returns a set of NFA states reachable from all states \( s \in T \) by \( \varepsilon \)-transitions
  - move(\( T \), \( a \)), for \( T \) a set of states in the NFA
    - Returns a set of NFA states to which there is a transition on \( a \) from some NFA state \( s \in T \)
- Build set of states (\( D \)) and transitions (\( D_{\text{trans}} \)) for the DFA.
Conversion Algorithm

Assume all states in NFA are unmarked initially.
Let \( S = \varepsilon\text{-closure}(\text{start state of NFA}) \).
Let \( D = \{S\} \).
while \( \exists \) an unmarked state \( T \in D \) do
   Mark \( T \);
   \( \forall \) input symbols \( a \) do
   \{ \( U = \varepsilon\text{-closure}(\text{move}(T, a)) \);
   if \( U \notin D \) then \{add unmarked \( U \) to \( D \);\}
   \( D_{\text{trans}}(T, a) = U \);
   \}
endwhile

Next Time

- Read ASU 4.1 & 4.2