Implementation of Lexical Analysis

Lecture 4

Written Assignments

- WA1 assigned today
- Due in one week
  - 11:59pm
  - Electronic hand-in

Tips on Building Large Systems

- KISS (Keep It Simple, Stupid!)
- Don't optimize prematurely
- Design systems that can be tested
- It is easier to modify a working system than to get a system working

Outline

- Specifying lexical structure using regular expressions
- Finite automata
  - Deterministic Finite Automata (DFAs)
  - Non-deterministic Finite Automata (NFAs)
- Implementation of regular expressions

  RegExp => NFA => DFA => Tables

Notation

- There is variation in regular expression notation
- Union: A | B ≡ A + B
- Option: A + ε ≡ A?
- Range: 'a'+'b'+…+'z' ≡ [a-z]
- Excluded range:

  complement of [a-z] ≡ [^a-z]

Regular Expressions in Lexical Specification

- Last lecture: a specification for the predicate

  $s \in L(R)$
- But a yes/no answer is not enough!
- Instead: partition the input into tokens
- We adapt regular expressions to this goal

Regular Expressions => Lexical Spec. (1)

1. Write a rexp for the lexemes of each token
   - Number = digit +
   - Keyword = 'if' + 'else' + …
   - Identifier = letter (letter + digit)*
   - OpenPar = '('
   - …

Regular Expressions => Lexical Spec. (2)

2. Construct R, matching all lexemes for all tokens

R = Keyword + Identifier + Number + …
$= R_1 + R_2 + \ldots$

Regular Expressions => Lexical Spec. (3)

3. Let input be $x_1 \ldots x_n$
   For $1 \le i \le n$ check
   $x_1 \ldots x_i \in L(R)$
4. If success, then we know that
   $x_1 \ldots x_i \in L(R_j)$ for some j
5. Remove $x_1 \ldots x_i$ from input and go to (3)

Ambiguities (1)

- There are ambiguities in the algorithm
- How much input is used? What if
  - $x_1 \ldots x_i \in L(R)$ and also
  - $x_1 \ldots x_K \in L(R)$
- Rule: Pick longest possible string in $L(R)$
  - The "maximal munch"

Ambiguities (2)

- Which token is used? What if
  - $x_1 \ldots x_i \in L(R_j)$ and also
  - $x_1 \ldots x_i \in L(R_k)$
- Rule: use rule listed first (j if $j < k$)
  - Treats "if" as a keyword, not an identifier

Error Handling

- What if
  No rule matches a prefix of input ?
- Problem: Can't just get stuck …
- Solution:
  - Write a rule matching all "bad" strings
  - Put it last (lowest priority)

Summary

- Regular expressions provide a concise notation for string patterns
- Use in lexical analysis requires small extensions
  - To resolve ambiguities
  - To handle errors
- Good algorithms known
  - Require only single pass over the input
  - Few operations per character (table lookup)

Finite Automata

- Regular expressions = specification
- Finite automata = implementation
- A finite automaton consists of
  - An input alphabet Σ
  - A set of states S
  - A start state n
  - A set of accepting states $F \subseteq S$
  - A set of transitions $state \rightarrow^{input} state$

Finite Automata

- Transition

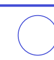
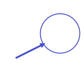
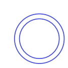
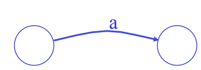
$$s_1 \rightarrow^{a} s_2$$

- Is read

  In state s_1 on input "a" go to state s_2

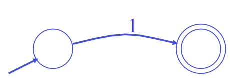
- If end of input and in accepting state => accept
- Otherwise => reject

Finite Automata State Graphs

- A state 
- The start state 
- An accepting state 
- A transition 

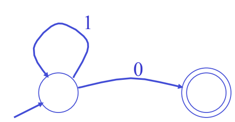
A Simple Example

- A finite automaton that accepts only "1"



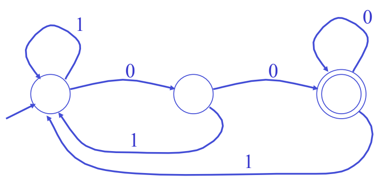
Another Simple Example

- A finite automaton accepting any number of 1's followed by a single 0
- Alphabet: {0,1}



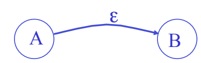
And Another Example

- Alphabet {0,1}
- What language does this recognize?



Epsilon Moves

- Another kind of transition: ε-moves



- Machine can move from state A to state B without reading input

Deterministic and Nondeterministic Automata

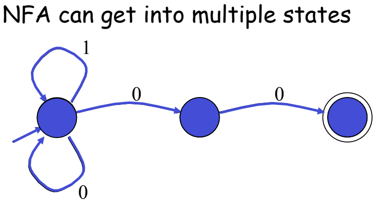
- Deterministic Finite Automata (DFA)
  - One transition per input per state
  - No ε-moves
- Nondeterministic Finite Automata (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have ε-moves

Execution of Finite Automata

- A DFA can take only one path through the state graph
  - Completely determined by input
- NFAs can choose
  - Whether to make ε-moves
  - Which of multiple transitions for a single input to take

Acceptance of NFAs

- An NFA can get into multiple states



- Input: 1 0 0

Rule: NFA accepts if it can get to a final state

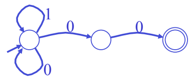
NFA vs. DFA (1)

- NFAs and DFAs recognize the same set of languages (regular languages)
- DFAs are faster to execute
  - There are no choices to consider

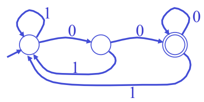
NFA vs. DFA (2)

- For a given language NFA can be simpler than DFA

NFA



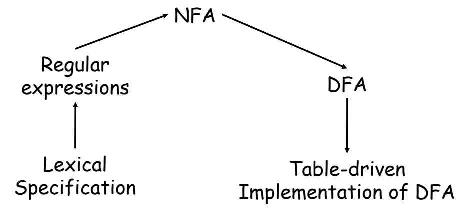
DFA



- DFA can be exponentially larger than NFA

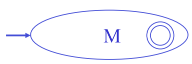
Regular Expressions to Finite Automata

- High-level sketch

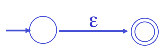


Regular Expressions to NFA (1)

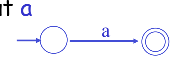
- For each kind of rexp, define an NFA
  - Notation: NFA for rexp M



- For ε

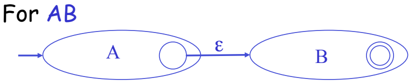


- For input a

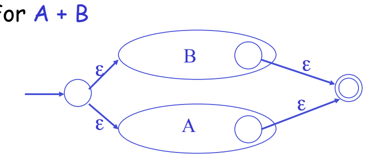


Regular Expressions to NFA (2)

- For AB

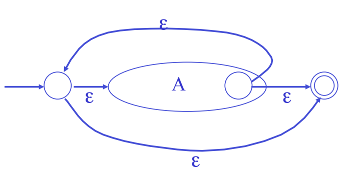


- For A + B



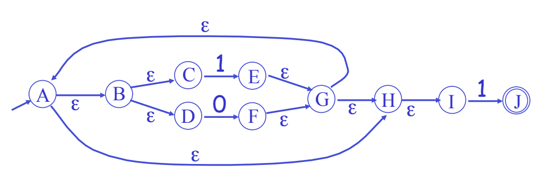
Regular Expressions to NFA (3)

- For A*



Example of RegExp -> NFA conversion

- Consider the regular expression
  (1+0)*1
- The NFA is



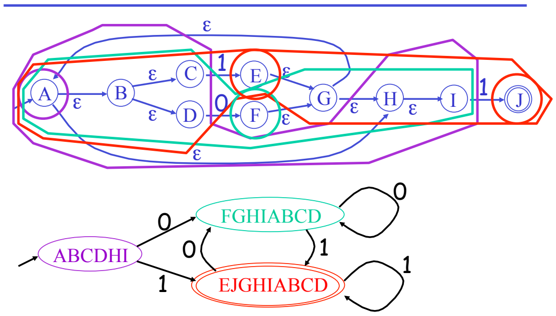
NFA to DFA: *The Trick*

- Simulate the NFA
- Each state of DFA
  = a non-empty subset of states of the NFA
- Start state
  = the set of NFA states reachable through ε-moves from NFA start state
- Add a transition $S \rightarrow^a S'$ to DFA iff
  - S' is the set of NFA states reachable from any state in S after seeing the input a, considering ε-moves as well

NFA to DFA. Remark

- An NFA may be in many states at any time
- How many different states ?
- If there are N states, the NFA must be in some subset of those N states
- How many subsets are there?
  - $2^N - 1$ = finitely many

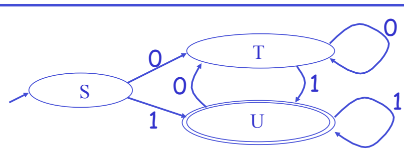
NFA -> DFA Example



Implementation

- A DFA can be implemented by a 2D table T
  - One dimension is "states"
  - Other dimension is "input symbol"
  - For every transition $S_i \rightarrow^a S_k$ define $T[i,a] = k$
- DFA "execution"
  - If in state S_i and input a, read $T[i,a] = k$ and skip to state S_k
  - Very efficient

Table Implementation of a DFA



	0	1
S	T	U
T	T	U
U	T	U

Implementation (Cont.)

- NFA -> DFA conversion is at the heart of tools such as flex
- But, DFAs can be huge
- In practice, flex-like tools trade off speed for space in the choice of NFA and DFA representations