Design of a Lexical-Analyzer Generator

Compiler Design Lexical Analysis
s.l. dr. ing. Ciprian-Bogdan Chirila
chirila@cs.upt.ro
http://www.cs.upt.ro/~chirila
Outline

• The Structure of the Generated Analyzer
• Pattern Matching Based on NFA’s
• DFA’s for Lexical Analyzers
• Implementing the Lookahead Operator
Objectives

- to present the architecture of Lex
- to discuss two approaches
  - NFA based
  - DFA based
    - implementation of Lex
The Structure of the Generated Lexical Analyzer

- fixed program that simulates an automaton
  - deterministic
  - nondeterministic
- transition table for the automaton
- functions that are passed directly through Lex to the output (we will see next)
- actions from the input program
  - as fragments of code
  - to be invoked at the appropriate time by the automaton simulator
Architecture of a Lexical Analyzer
Generated by Lex
The Generation Process

- each regular expression pattern is transformed into NFA
- all NFAs are combined into one
  - new $\varepsilon$-transitions are added to NFAs $N_i$ for pattern $p_i$
Example
Example

- patterns
  - a \{\text{action } A_1 \text{ for pattern } p_1\}
  - abb \{\text{action } A_2 \text{ for pattern } p_2\}
  - a^*b^+ \{\text{action } A_3 \text{ for pattern } p_3\}

- when several prefixes on the input matches multiple patterns
  - always prefer a longer prefix to a shorter prefix
  - if the longest possible prefix matches multiple patterns choose the pattern listed first
    - the lexeme “abb” is taken by the second rule
Conflict Resolution

- the three patterns present some conflicts
- **aab** matches p2 and p3
  - we consider it a lexeme for p2
  - p2 is listed **above** p3
- **aabbbb**…
  - we take the **longest** lexeme until another a is reached
  - we will report the lexeme from the initial a followed by as many b as there are
Example
Pattern Matching Based on NFA’s

- NFA simulation algorithm
  
  \[ S = \varepsilon - \text{closure}(s_0); \]
  
  \[ c = \text{nextChar}(); \]
  
  \[ \text{while}(c \neq \text{eof}) \]
  
  \[ \{ \]
  
  \[ S = \varepsilon - \text{enclosure}(\text{move}(S, c)); \]
  
  \[ c = \text{nextChar}(); \]
  
  \[ \} \]
  
  \[ \text{if}(S \cap F \neq \emptyset) \text{ return } \text{“yes”; } \]
  
  \[ \text{else return } \text{“no”; } \]
Example input a a b a
Example input a a b a
Example input a a b a
Example input a a b a

- pattern a*b+ was found !!!
DFAs Architecture for Lexical Analyzers

- to **convert** NFA for all patterns into DFA
  - by using the **subset construction algorithm**
- within each DFA state having one or more NFA accepting states
  - to **determine** the **first pattern** whose accepting state is represented
  - to **make** that **pattern** the **output** of the DFA state
The Subset Construction Algorithm

while(there is an unmarked state T in Dstates)
{
    mark T;
    for(each input symbol a)
    {
        U=\varepsilon\text{-closure}(\text{move}(T,a));
        if (U is not in Dstates)
            add U as unmarked state to Dstates;
        Dtran[T,a]=U;
    }
}
NFA Example
NFA to DFA Example
DFA Simulation Example \textcolor{red}{a b b b a}
DFA Simulation Example a b b a
DFA Simulation Example a b b a
Dead States in DFA’s

- the automaton not quite a DFA
  - no transitions on every state x every input
- we have omitted
  - transitions to the dead state Ø
  - from the dead state Ø to itself
Bibliography