Overview

- Generating a lexical analyzer
  - generic methods
  - specific tool lex

Token description

- (f)lex: scanner generator for UNIX
  - token description → C code

  - format of the lex input file:
    
    `definitions` ← regular descriptions
    
    `rules` ← regular expressions + actions
    
    `user code` ← auxiliary C-code

Lex description to recognize integers

- an integer is a non-zero sequence of digits optionally followed by a letter denoting the base class (b for binary and o for octal).

  - base → [bo]
  
  - integer → digit + base?

  - rule = expr + action

  - {} signal application of a description

Lex resulting C-code

- char yytext[]; /* token representation */
- int yylex(void); /* returns type of next token */

  - wrapper function to add token attributes

automatic generation

- program text
  - lexical analysis
    - tokens
  - syntax analysis
    - AST
  - context handling
    - annotated AST

finite state automation

S0

S1

S2

S3

S4

S0

S1

S2

S3

S4
**Finite-state automaton**

- Recognize input character by character
- Transfer between states

- FSA
  - Initial state S0
  - set of accepting states
  - transition function: State x Char → State

---

**FSA examples**

- integral_number → [0-9]+
- fixed_point_number → [0-9]* '.' [0-9]+

---

**Concurrent recognition**

- integral_number → [0-9]+
- fixed_point_number → [0-9]* '.' [0-9]+  
- recognize both tokens in one pass

---

**Concurrent recognition**

- integral_number → [0-9]+  
- fixed_point_number → [0-9]* '.' [0-9]+  
- naïve approach: merge initial states

---

**Concurrent recognition**

- integral_number → [0-9]+  
- fixed_point_number → [0-9]* '.' [0-9]+  
- correct approach: share common prefix transitions

---

**FSA implementation: transition table**

- concurrent recognition of integers and fixed point numbers

<table>
<thead>
<tr>
<th>state</th>
<th>character</th>
<th>recognized token</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>-</td>
</tr>
<tr>
<td>S3</td>
<td>S3</td>
<td>-</td>
</tr>
</tbody>
</table>
FSA exercise (6 min.)

- draw an FSA to recognize integers
  
  \[ \text{base} \rightarrow [bo] \]
  \[ \text{integer} \rightarrow \text{digit}^+ \text{ base}^? \]

- draw an FSA to recognize the regular expression \((a|b)^*bab\)

Answers

Automatic generation: description \(\rightarrow\) FSA

- start with initial set \((S_0)\) of all token descriptions to be recognized

- for each character \((ch)\)
  
  * find the set \((S_{ch})\) of descriptions that can start with \(ch\)
  * extend the FSA with transition \((S_0, ch, S_{ch})\)

- repeat adding transitions (to \(S_{ch}\)) until no new set is generated

Dotted items

- keeping track of matched characters in a token description: \(T \rightarrow R\)

Types of dotted items

- shift item: dot in front of a basic pattern
  
  * if \(\rightarrow \star \text{ if}\)
  * if \(\rightarrow \star \text{ if}\)
  * identifier \(\rightarrow \star \text{ [a-z]} \text{ [a-z0-9]}^*\)

- reduce item: dot at the end
  
  * if \(\rightarrow \star \text{ if}\)
  * identifier \(\rightarrow \star \text{ [a-z]} \text{ [a-z0-9]}^*\)

- non-basic item: dot in front of repeated pattern or parenthesis
  
  * identifier \(\rightarrow \star \text{ [a-z]} \text{ [a-z0-9]}^*\)

Character moves

- \(T \rightarrow \alpha \cdot c \beta\)
  
  * if \(\rightarrow c\)
  * \(T \rightarrow \alpha \cdot [\text{class}] \beta\)
  * \(T \rightarrow \alpha \cdot . \beta\)
  
  \(T \rightarrow \alpha \cdot c \beta\)
  
  \(c \in \text{class}\)
  
  \(T \rightarrow \alpha \cdot [\text{class}] \cdot \beta\)
  
  \(T \rightarrow \alpha \cdot . \beta\)
### ε moves

- $T \rightarrow \alpha \bullet (R)? \beta \Rightarrow T \rightarrow \alpha (R)? \bullet \beta$
- $T \rightarrow \alpha (R)? \bullet \beta \Rightarrow T \rightarrow \alpha (R)? \bullet \beta$
- $T \rightarrow \alpha \bullet (R)* \beta \Rightarrow T \rightarrow \alpha (R)* \bullet \beta$
- $T \rightarrow \alpha (R)* \bullet \beta \Rightarrow T \rightarrow \alpha (R)* \bullet \beta$
- $T \rightarrow \alpha (R)* \bullet \beta \Rightarrow T \rightarrow \alpha (R)* \bullet \beta$
- $T \rightarrow \alpha (R)+ \bullet \beta \Rightarrow T \rightarrow \alpha (R)+ \bullet \beta$
- $T \rightarrow \alpha (R)+ \bullet \beta \Rightarrow T \rightarrow \alpha (R)+ \bullet \beta$
- $T \rightarrow \alpha (R)+ \bullet \beta \Rightarrow T \rightarrow \alpha (R)+ \bullet \beta$
- $T \rightarrow \alpha (R)+ \bullet \beta \Rightarrow T \rightarrow \alpha (R)+ \bullet \beta$
- $T \rightarrow \alpha (R)+ \bullet \beta \Rightarrow T \rightarrow \alpha (R)+ \bullet \beta$

### FSA construction

- A state corresponds to a set of basic items
- A character move yields a new set
- Expand non-basic items into basic items using ε moves
- See if the resulting set was produced before, if not introduce a new state
- Add transition

### Example FSA construction

- Tokens
  - Integer: $I \rightarrow (D)^+$
  - Fixed-point: $F \rightarrow (D)^+ \bullet \cdot (D)^+$
- Initial state

- Draw the FSA (with item sets) for recognizing an identifier:

  \[
  \text{identifier} \rightarrow \text{letter} \ (\text{letter_or_digit_or_und}* \text{letter_or_digit}^+)?
  \]

- Extend the above FSA to recognize the keyword ‘if’ as well.

  \[
  \text{if} \rightarrow 'i' 'f'
  \]
Answers

Transition table compression

<table>
<thead>
<tr>
<th>state</th>
<th>character</th>
<th>recognized token</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S1</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S2</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S3</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S4</td>
<td>T</td>
<td>S1, S3, S1, S1</td>
</tr>
</tbody>
</table>

• redundant rows
• empty transitions

<table>
<thead>
<tr>
<th>state</th>
<th>character</th>
<th>recognized token</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S1</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S2</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S3</td>
<td>T</td>
<td>S1, S3</td>
</tr>
<tr>
<td>S4</td>
<td>T</td>
<td>S1, S3, S1, S1</td>
</tr>
</tbody>
</table>

• redundant rows
• empty transitions

Summary: generating a lexical analyzer

• tool: lex
  • token descriptions + actions
  • wrapper interface
• FSA construction
  • dotted items
  • character moves
  • $\varepsilon$ moves

Homework

• study sections 2.1.10 – 2.1.12
  • lexical identification of tokens
  • symbol tables
  • macro processing
• print handout lecture 3 [blackboard]
• find a partner for the “practicum”
• register your group
• send e-mail to koen@pds.twi.tudelft.nl