

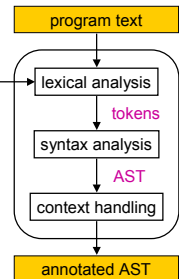
Compiler construction in4020 – lecture 2

Koen Langendoen

Delft University of Technology
The Netherlands

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

```

%%
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

```

%{
#include    "lex.h"
}%

base      [bo]
digit     [0-9]

%%
{digit}+ {base}? {return INTEGER;}
%%
    
```

Lex resulting C-code

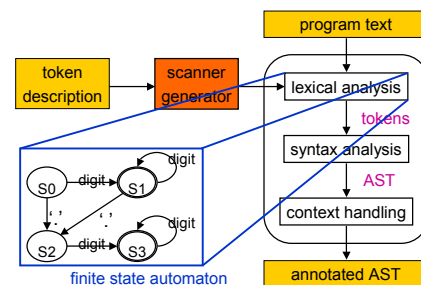
- `char yytext[]; /* token representation */`
- `int yylex(void); /* returns type of next token */`

- wrapper function to add token attributes

```

%%
\n        {line_number++;}
%%
void get_next_token(void) {
    Token.class = yylex();
    if (Token.class == 0) {
        Token.class = EOF;
        Token.repr = "<EOF>";
        return;
    }
    Token.pos.line_number = line_number;
    Token.repr = strdup(yytext);
}
    
```

automatic generation

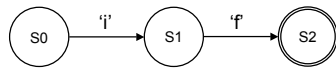


Finite-state automaton

- Recognize input character by character
- Transfer between states

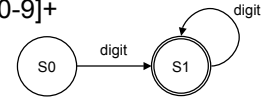
FSA

- Initial state S0
- set of accepting states
- transition function: State x Char \rightarrow State



FSA examples

- integral_number $\rightarrow [0-9]^+$

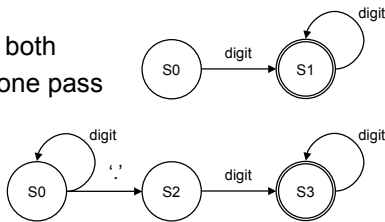


- fixed_point_number $\rightarrow [0-9]^* '.' [0-9]^+$



Concurrent recognition

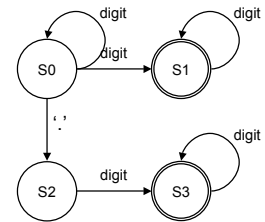
- integral_number $\rightarrow [0-9]^+$
- fixed_point_number $\rightarrow [0-9]^* '.' [0-9]^+$
- recognize both tokens in one pass



Concurrent recognition

- integral_number $\rightarrow [0-9]^+$
- fixed_point_number $\rightarrow [0-9]^* '.' [0-9]^+$

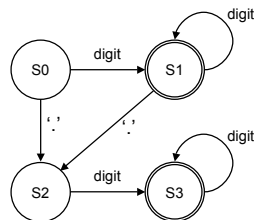
- naïve approach: merge initial states



Concurrent recognition

- integral_number $\rightarrow [0-9]^+$
- fixed_point_number $\rightarrow [0-9]^* '.' [0-9]^+$

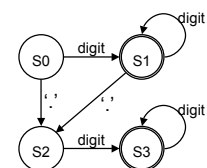
- correct approach: share common prefix transitions



FSA implementation: transition table

- concurrent recognition of integers and fixed point numbers

state	character			recognized token
	digit	dot	other	
S0	S1	S2	-	
S1	S1	S2	-	integer
S2	S3	-	-	
S3	S3	-	-	fixed point



FSA exercise (6 min.)

- draw an FSA to recognize integers
 $base \rightarrow [bo]$
 $integer \rightarrow digit+ base?$
- draw an FSA to recognize the regular expression $(a|b)^*bab$

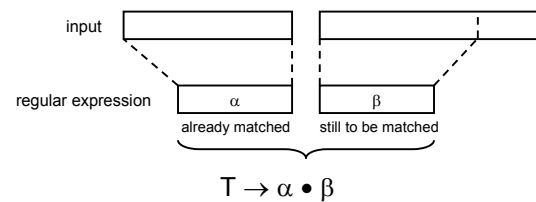
Answers

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

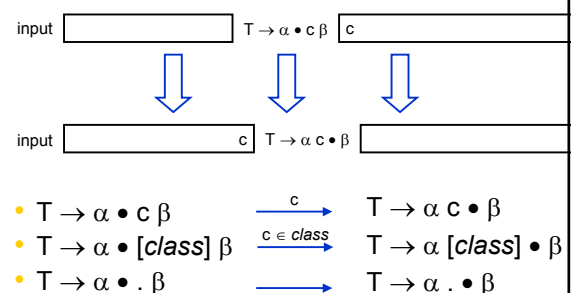
- keeping track of matched characters in a token description: $T \rightarrow R$



Types of dotted items

- shift item:** dot in front of a basic pattern
 - if $\rightarrow \bullet 'i' 'f'$
 - if $\rightarrow 'i' \bullet 'f'$
 - identifier $\rightarrow \bullet [a-z] [a-z0-9]^*$
- reduce item:** dot at the end
 - if $\rightarrow 'i' 'f' \bullet$
 - identifier $\rightarrow [a-z] [a-z0-9]^* \bullet$
- non-basic item:** dot in front of repeated pattern or parenthesis
 - identifier $\rightarrow [a-z] \bullet [a-z0-9]^*$

Character moves



ϵ moves

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

ϵ moves

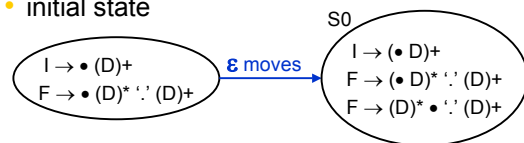
$T \rightarrow \alpha \bullet (R)^+ \beta$	\Rightarrow	$T \rightarrow \alpha (\bullet R)^+ \beta$
$T \rightarrow \alpha (R \bullet)^+ \beta$	\Rightarrow	$T \rightarrow \alpha (R)^+ \bullet \beta$
		$T \rightarrow \alpha (\bullet R)^+ \beta$
$T \rightarrow \alpha \bullet (R_1 R_2 \dots) \beta$	\Rightarrow	$T \rightarrow \alpha (\bullet R_1 R_2 \dots) \beta$
		$T \rightarrow \alpha (R_1 \bullet R_2 \dots) \beta$
		...
$T \rightarrow \alpha (R_1 \bullet R_2 \dots) \beta$	\Rightarrow	$T \rightarrow \alpha (R_1 R_2 \dots) \bullet \beta$
...		...

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

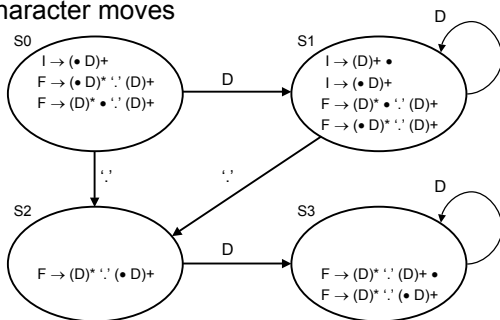
Example FSA construction

- tokens
 - integer:** $I \rightarrow (D)^+$
 - fixed-point:** $F \rightarrow (D)^* \cdot (D)^+$
- initial state



Example FSA construction

- character moves



Exercise FSA construction (7 min.)

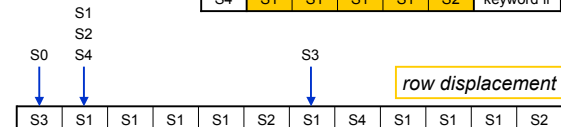
- draw the FSA (with item sets) for recognizing an identifier:
 - identifier** \rightarrow letter (letter_or_digit_or_und* letter_or_digit+)?
- extend the above FSA to recognize the keyword 'if' as well.
 - if** \rightarrow 'i' 'f'

Answers

Transition table compression

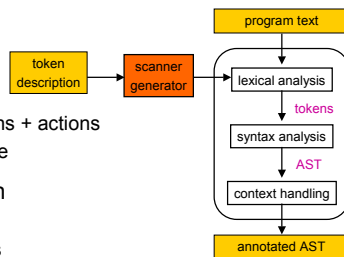
- redundant rows
- empty transitions

state	character					recognized token
	T	f	L	D	U	
S0	S3	S1	S1	-	-	
S1	S1	S1	S1	S1	S2	identifier
S2	S1	S1	S1	S1	S2	
S3	S1	S4	S1	S1	S2	
S4	S1	S1	S1	S1	S2	keyword if



Summary: generating a lexical analyzer

- tool: **lex**
 - token descriptions + actions
 - wrapper interface
- FSA construction
 - dotted items
 - character moves
 - ϵ 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