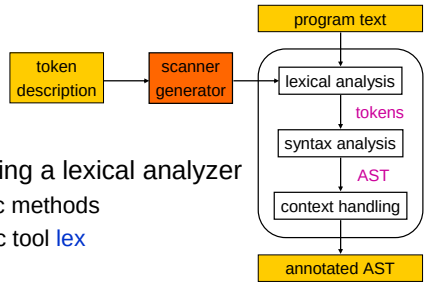
Compiler construction in4303 – lecture 2

Automatic lexical analysis

Chapter 2.1.6 - 2.1.13

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

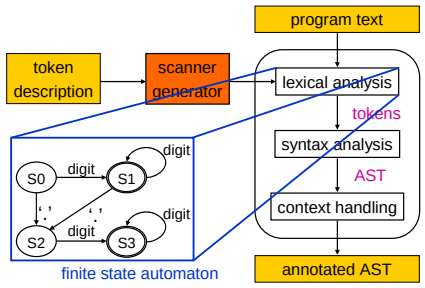
```
%{
#include    "lex.h"
%}

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

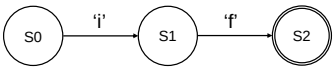
```
%%
...
\n      line_number++;
%%
void get_next_token(void) {
    Token.class = yylex();
    if (Token.class == 0) {
        Token.class = EOF;
        Token.repr = "<EOF>";
        return;
    }
    Token.pos.line_number = line_number;
    Token.repr = strdup(yytext);
}
```

automatic generation



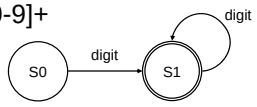
Finite-state automaton

- Recognize input character by character
- Transfer between states
- FSA
  - Initial state S0
  - set of accepting states
  - transition function: State x Char → State

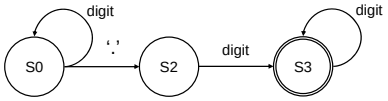


FSA examples

- integral_number → [0-9]+

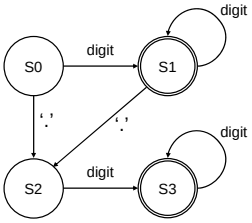


- fixed_point_number → [0-9]* '.' [0-9]+



Concurrent recognition

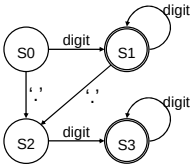
- integral_number → [0-9]+
- fixed_point_number → [0-9]* '.' [0-9]+
- correct approach: share common prefix transitions



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 → [bo]
  integer → *digit+ base*?

- draw an FSA to recognize the regular expression [ab]*bab

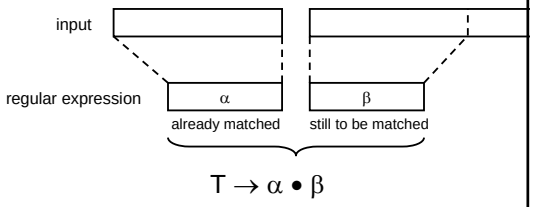
Answers

Automatic generation: description → FSA

- start with initial set (S0) 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 (S0,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 → R

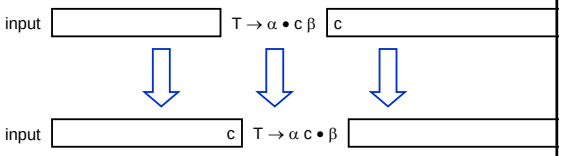


$T \rightarrow \alpha \bullet \beta$

Types of dotted items

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

Character moves



- $T \rightarrow \alpha \bullet c\ \beta$ —c→ $T \rightarrow \alpha\ c \bullet \beta$
- $T \rightarrow \alpha \bullet [class]\ \beta$ —c ∈ class→ $T \rightarrow \alpha\ [class] \bullet \beta$
- $T \rightarrow \alpha \bullet .\ \beta$ → $T \rightarrow \alpha\ . \bullet \beta$

ε moves

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

ε moves

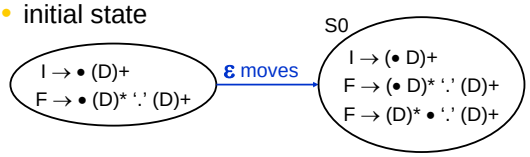
$T \rightarrow \alpha \bullet (R)+\ \beta$	⇒	$T \rightarrow \alpha\ (\bullet R)+\ \beta$
$T \rightarrow \alpha\ (R \bullet)+\ \beta$	⇒	$T \rightarrow \alpha\ (R)+ \bullet \beta$
		$T \rightarrow \alpha\ (\bullet R)+\ \beta$
$T \rightarrow \alpha \bullet (R_1\|R_2\|\ldots)\ \beta$	⇒	$T \rightarrow \alpha\ (\bullet R_1\|R_2\|\ldots)\ \beta$
		$T \rightarrow \alpha\ (R_1\| \bullet R_2\|\ldots)\ \beta$
		…
$T \rightarrow \alpha\ (R_1 \bullet \|R_2\|\ldots)\ \beta$	⇒	$T \rightarrow \alpha\ (R_1\|R_2\|\ldots) \bullet \beta$
…	…	…

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

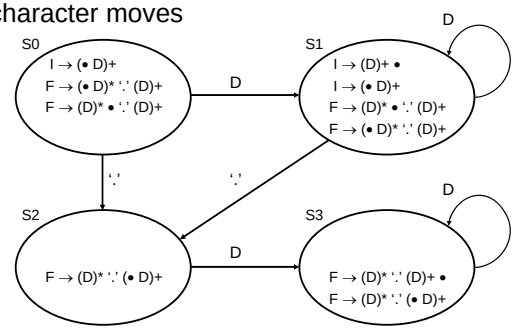
Example FSA construction

- tokens
  - integer: I → (D)+
  - fixed-point: F → (D)* '.' (D)+
- initial state



Example FSA construction

- character moves



Exercise FSA construction (7 min.)

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

  identifier → *letter* (*letter_or_digit_or_und** *letter_or_digit)?*

- extend the above FSA to recognize the keyword 'if' as well.

  if → *'i' 'f'*

Answers

Transition table compression

- redundant rows
- empty transitions

state	character					recognized token
	'i'	'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	identifier
S4	S1	S1	S1	S1	S2	keyword if