COMP3131/9102: Programming Languages and Compilers

Jingling Xue

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
`http://www.cse.unsw.edu.au/~cs3131`
`http://www.cse.unsw.edu.au/~cs9102`

Week 3 (1st Lecture): Top-Down Parsing (Revisited)

- Feedback for Assignment 1 (see the feedback video online)
- Revisit First and Follow sets
- Revisit Top-Down Parsing
- Formal Grammars
- Equivalence between Regular Grammars and FAs

A Simple Tool for Computing First and Follow Sets

`https://gist.github.com/DmitrySoshnikov/924ceefb1784b30c5ca6`

LL(k) Grammar and Parsing

- A grammar is LL(k) if it can be parsed deterministically using k tokens of lookahead
- A formal definition for LL(k) grammars can be found in Grune and Jacobs' book

  `https://dickgrune.com/Books/PTAPG_1st_Edition/`
- Grammar 1 in Slide 171 is not LL(k) for any k!
- However, Grammar 2 in Slide 171 is LL(1)
- Only a understanding of LL(1) is required this year

The VC Grammar Is Not LL(1)

- $Program$: common prefix in its production right-hand sides
- A lots of left-recursive productions

Must eliminate both parsing conflicts to write your recogniser for Assignment 2

Formal Grammar

A grammar G is a quadruple (V_T, V_N, S, P), where

- V_T: a finite set of terminal symbols or tokens
- V_N: a finite set of nonterminal symbols ($V_T \cap V_N = \emptyset$)
- S: a unique start symbol ($S \in N$)
- P: a finite set of rules or productions of the form:

$$\alpha \to \beta \qquad\qquad (\alpha \neq \epsilon)$$

  - α is a string of one or more terminals and nonterminals
  - β is a string of zero or more terminals and nonterminals

Chomsky's Hierarchy

Depending on $\alpha \rightarrow \beta$, four types of grammars distinguished:

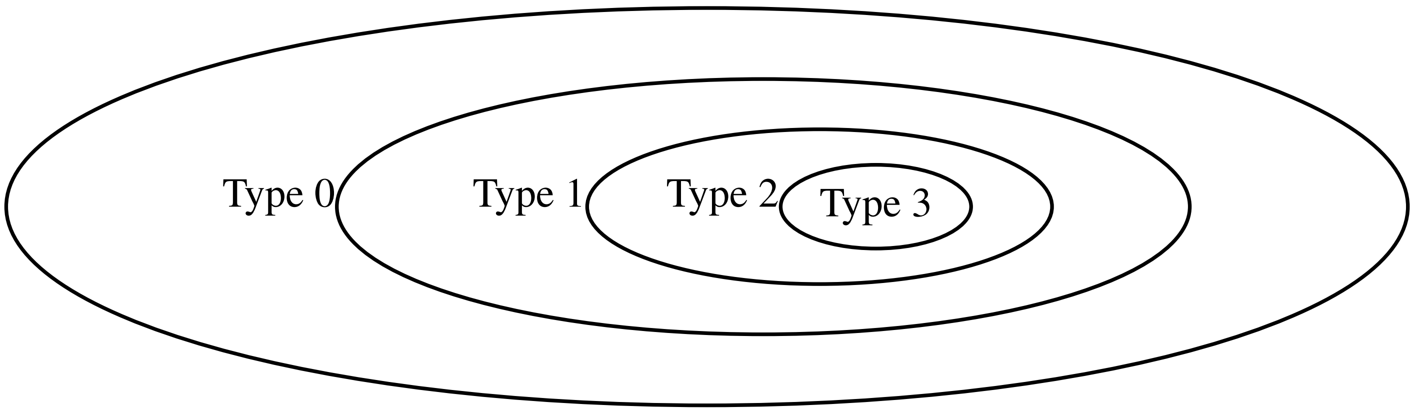
GRAMMAR	KNOWN AS	DEFINITION	MACHINE
Type 0	phrase-structure grammar	$\alpha \neq \epsilon$	Turing machine
Type 1	context-sensitive grammar CSGs	$\|\alpha\| \leq \|\beta\|$	linear bounded automaton
Type 2	context-free grammar CFGs	$A \rightarrow \alpha$	stack automaton
Type 3	right-linear grammar regular grammars	$A \rightarrow a \mid aB$	finite automaton

Note:

- a is a terminal.
- regular grammars can also be specified by left-linear grammars:

$$A \rightarrow a \mid Ba$$

Relationships between the Four Types of Languages



- Type k language is a proper subset of Type $k-1$ language.
- The existence of a Type 0 language is proved:

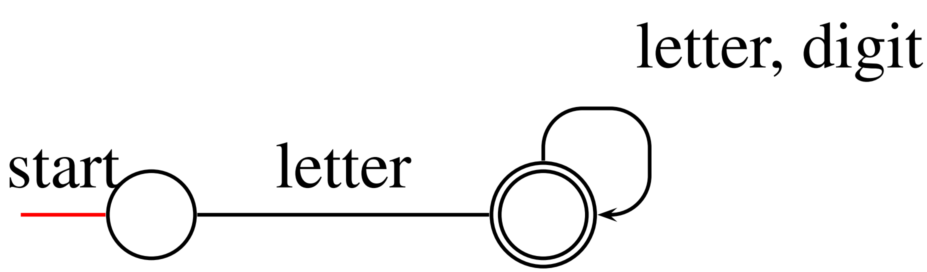
  page 228, J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

Regular Expressions, Regular Grammars and Finite Automata

- All three are equivalent:
- Example:
  - Regular expression: $[A-Za-z\_][A-Za-z0-9\_]^*$
  - Regular grammar:

    identifier `->` *letter* `|` *identifier letter* `|` *identifier digit*
    letter `-> A|B| ... |Z|a|b| ... |z|_`
    digit `-> 0|1| ... |9`

  - DFA:



Limitations of Regular Grammars

- Cannot generate nested constructs
- The following language is not regular

$$L = \{a^n b^n \mid n \geqslant 0\}$$

- But L is context-free: $S \rightarrow \epsilon \mid aSb$
- Regular grammars (expressions) powerful enough for specifying tokens, which are not nested
- By replacing "a" and "b" with "(" and ")", the following

$$L = \{(^n)^n \mid n \geqslant 0\}$$

  is not regular
- Formal proof: Pages 180 – 181 of Red / $4.2.7 of Purple
- Regular grammars (finite automata) cannot count

Limitations of CFGs

- CFLs only include a subset of all languages
- Examples of non-CFL constructs:
  - An abstraction of variable declaration before use:

    $$L_1 = \{wcw \mid \text{w is in } (a|b)^*\}$$

    where the 1st w represents a declaration and the 2nd its use
  - a method called with the right number of arguments:

    $$L_2 = \{a^n b^m c^n d^m \mid n \geqslant 1, m \geqslant 1\}$$

    where a^n and b^m represent formal parameter lists in two methods with n and m arguments, respectively, and c^n and d^m represent actual parameter lists in two calls to the two methods.
- Can count two but not three:

  $$L_3 = \{a^n b^n c^n \mid n \geqslant 0\}$$

Limitations of CFGs (Cont'd)

- L_3 is not context-free
  - The language:

$$L_3 \;=\; \{a^n b^n c^n \mid n \geqslant 0\}$$

  - The grammar:

- A Context-Sensitive Grammar (CSG) (that is not a CFG) for L_3:

CSG:			A derivation for $aabbcc$		
S	$\rightarrow$	$aSBC$	S	$\Longrightarrow$	$aSBC$
S	$\rightarrow$	abC		$\Longrightarrow$	$aabCBC$
CB	$\rightarrow$	BC		$\Longrightarrow$	$aabBCC$
bB	$\rightarrow$	bb		$\Longrightarrow$	$aabbCC$
bC	$\rightarrow$	bc		$\Longrightarrow$	$aabbcC$
cC	$\rightarrow$	cc		$\Longrightarrow$	$aabbcc$

Why CFGs in Parser Construction?

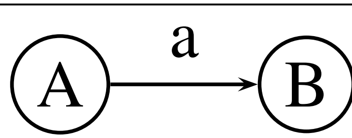
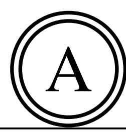
- Types 0 and 1 are less understood, no simple ways of constructing parsers for them, and parsers for these languages are slow
- Type 3 cannot define recursive language constructs
- Type 2 – context-free grammars (CFGs):
  - Easily related to the structure of the language; productions give us a good idea of what to expect in the language
  - Close relationships between the productions and the corresponding computations, which is the basis of syntax-directed translation
  - Efficient parsers can be built automatically from CFGs

Equivalence between Regular Grammars and FAs

- Week 1 (2nd Lecture): the equivalence among REs and FAs
- Slides 222 – 227: NFAs $\equiv$ Regular Grammars

Converting NFAs to Right-Linear Grammars

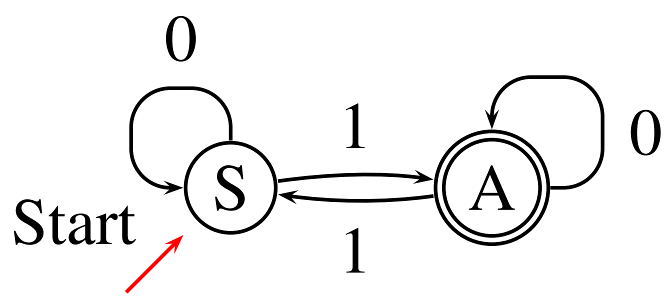
- The alphabet: the same
- For each state in the NFA, create a nonterminal with the same name.
- The start state will be the start symbol
- Then

TRANSITION		PRODUCTION
	$\Longrightarrow$	$A \rightarrow aB$
	$\Longrightarrow$	$A \rightarrow \epsilon$

where $a \in \sum$ or $a = \epsilon$

Example 1

- The DFA:

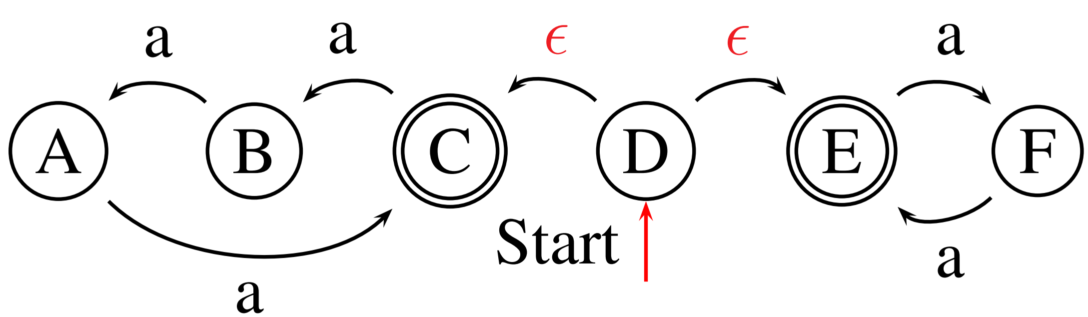


- The grammar:

$$
\begin{array}{rcl}
S & \rightarrow & 0S \\
S & \rightarrow & 1A \\
A & \rightarrow & 0A \\
A & \rightarrow & 1S \\
A & \rightarrow & \epsilon
\end{array}
$$

Example 2

- The NFA:



- The grammar:

$$
\begin{array}{lllllll}
D & \rightarrow & C & \quad & A & \rightarrow & aC \\
D & \rightarrow & E & \quad & E & \rightarrow & aF \\
C & \rightarrow & aB & \quad & E & \rightarrow & \epsilon \\
C & \rightarrow & \epsilon & \quad & F & \rightarrow & aE \\
B & \rightarrow & aA & & & &
\end{array}
$$

Converting Right-Linear Grammars to NFAs

- The alphabet: the same
- For each nonterminal, create a state in the NFA with the same name. The start symbol will be the start state
- Add one new state and make it the only final state $\mathcal{F}$
- Then

PRODUCTION		TRANSITION	
$A \rightarrow aB$	$\Longrightarrow$	$A \xrightarrow{a} B$	$T(A, a) = B$
$A \rightarrow a$	$\Longrightarrow$	$A \xrightarrow{a} \mathcal{F}$	$T(A, a) = \mathcal{F}$

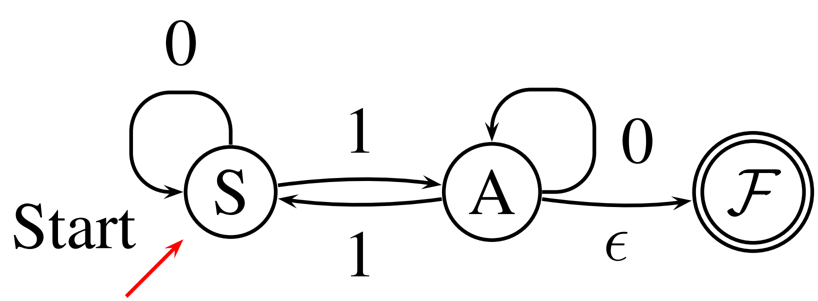
where $a \in \sum$ or $a = \epsilon$

Example 1

- The grammar:

$$
\begin{array}{lcl}
S & \rightarrow & 0S \\
S & \rightarrow & 1A \\
A & \rightarrow & 0A \\
A & \rightarrow & 1S \\
A & \rightarrow & \epsilon
\end{array}
$$

- The NFA:



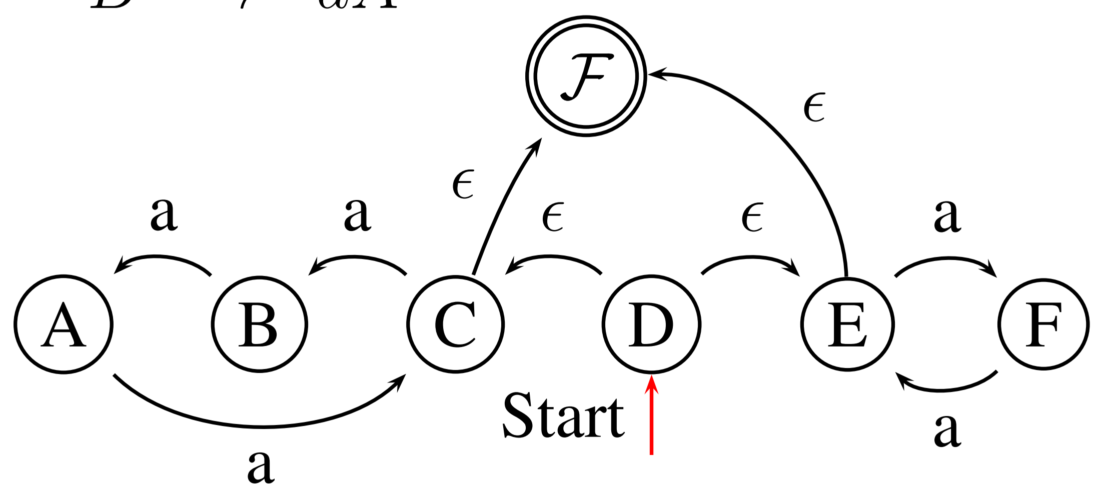
- This NFA accepts the same language as the one in Slide 223

Example 2

- The grammar:

$$
\begin{array}{lcl lcl}
D & \to & C & A & \to & aC \\
D & \to & E & E & \to & aF \\
C & \to & aB & E & \to & \epsilon \\
C & \to & \epsilon & F & \to & aE \\
B & \to & aA & & &
\end{array}
$$

- The NFA:



- This NFA accepts the same language as the NFA in Slide 224

Reading

- Many online material on computing First and Follow sets
- The Dragon textbook

Next Class: Abstract Syntax Trees (Preparing You for Assignment 3)