

FIT2014 Theory of Computation

Assignment Project Exam Help

Lecture 15

Parsing

<https://powcoder.com>

slides by Graham Farr

Add WeChat powcoder

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

Warning

This material has been reproduced and communicated to you by or on behalf of Monash University
in accordance with s113P of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act.

Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Assignment Project Exam Help

- ▶ Concepts and definitions
- ▶ Examples
- ▶ Shift-reduce parser
- ▶ Lex & Yacc

<https://powcoder.com>

Add WeChat powcoder

Parsing

Suppose you have a Context Free Grammar, and a string of letters.

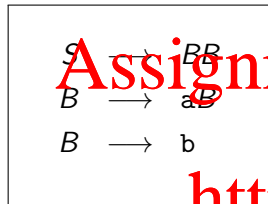
Parsing: determining whether the string

- ▶ is a word in the language, and if it is,
- ▶ finding a parse tree, or a derivation, for it.

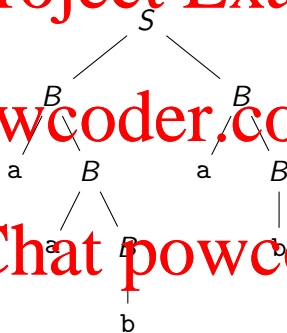
Parser: a program that does this.

- ▶ Two main types:
 - ▶ Top-down parsers
 - ▶ Bottom-up parsers
 - ▶ reduce the string to the Start symbol
 - ▶ repeatedly apply production rules in reverse

a^*ba^*b



Input string: aabab



Assignment Project Exam Help

<https://powcoder.com>

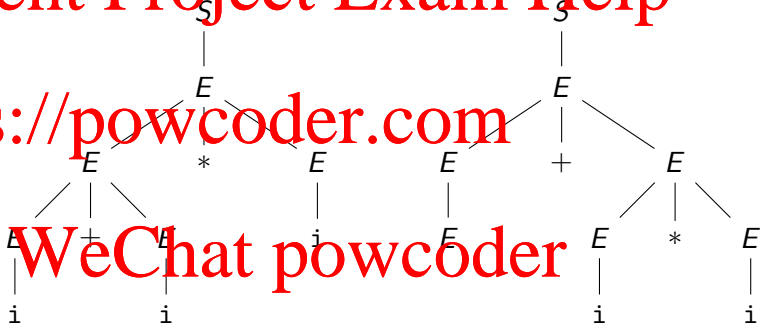
Add WeChat powcoder

Plus-Times-A

Input string: $i + i * i$

1. $S \rightarrow E$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow i$

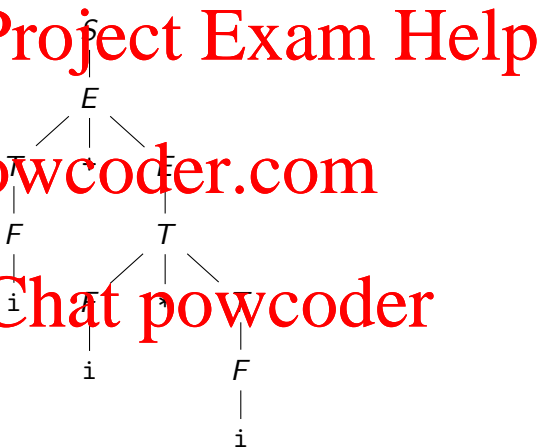
This grammar
is *ambiguous*.



Two parse trees

Input string: $i + i * i$

$S \rightarrow E$
 $E \rightarrow T + E$
 $E \rightarrow T$
 $T \rightarrow F * T$
 $T \rightarrow F$
 $F \rightarrow i$



- ▶ bottom-up parser
- ▶ scans input Left to Right
- ▶ constructs a Rightmost derivation in reverse
- ▶ implemented using a Deterministic Pushdown Automaton (DPDA)
- ▶ Not all CFGs have such a parser: $DCFL \neq CFL$.
- ▶ We'll look at one type of LR parser:
shift-reduce parser

<https://powcoder.com>

Add WeChat powcoder

Shift-reduce Parser

This is a particular type of LR Parser.

It has:

- ▶ a **stack**: terminals and non-terminals processed so far.
 - ▶ Initially: empty.
- ▶ a **buffer**: the rest of the input string (yet to be processed).
 - ▶ Initially: contains the entire input string.

Repeatedly ...

- ▶ **Shift** input letters onto the **stack**, OR
- ▶ When a string of top-most **stack** symbols equal the right-hand side of a production rule:
 - ▶ **Reduce** that string, i.e., use production rule in reverse

...until **Stack** only has Start symbol, and **buffer** is empty.

a^*ba^*b

Input: abb

Assignment Project Exam Help


1. $S \rightarrow BB$

2. $B \rightarrow aB$


3. $B \rightarrow b$

<https://powcoder.com>

Add WeChat powcoder

 abb shift

 bb shift

 b reduce, (3)

 b reduce, (2)

 b shift

 reduce, (3)

 reduce, (1)

 ACCEPT

Input: $i+i*i$

- | | | | | |
|----|-----------------------|---------|-------------|-------------|
| 1. | $S \rightarrow E$ | $i+i*i$ | shift | |
| 2. | $E \rightarrow E + E$ | $i+i+i$ | reduce, (4) | |
| 3. | $E \rightarrow E * E$ | E | $+i*i$ | shift |
| 4. | $E \rightarrow E$ | $E+$ | $i*i$ | shift |
| | | $E+i$ | $*i$ | reduce, (4) |
| | | $E+E$ | $*i$ | |

Add WeChat powcoder

To shift,
or to reduce?

Input: $i+i*i$

1. $S \rightarrow E$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow E$

	$i+i*i$	shift
	$i+i+i$	reduce, (4)
	$E+i+i$	shift
	$E+i*i$	shift
	$E+i*i$	reduce, (4)
	$E+E*i$	shift
	$E+E*i$	shift
	$E+E*i$	reduce, (4)
	$E+E*i$	reduce, (3)
	$E+E$	reduce, (2)
	E	reduce, (1)
	S	ACCEPT

<https://powcoder.com>

Add WeChat powcoder

1. $S \rightarrow E$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow i$

	$i+i*i$	shift
i	$+i*i$	reduce, (4)
E	$+i*i$	shift
$E+$	$i*i$	shift
$E+i$	$*i$	reduce, (4)
$E+E$	$*i$	

<https://powcoder.com>

Add WeChat **powcoder**

To **shift**,
or to **reduce**?

Plus-Times-A

Input: $i+i*i$

1. $S \rightarrow E$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow i$

	$i+i*i$	shift
i	$+i*i$	reduce, (4)
E	$+i*i$	shift
$E+$	$i*i$	shift
$E+i$	$*i$	reduce, (4)
$E+E$	$*i$	reduce, (2)
E	$*i$	shift
$E*$	i	shift
$E*i$		reduce, (4)
$E * E$		reduce, (3)
E		reduce, (1)
S		ACCEPT

→ shift-reduce conflict

Also:

reduce-reduce conflict: letters on top of stack correspond to > one production rule.

Unix/Linux tools

Yacc

- ▶ **Y**et **A**nother **C**ompiler-**C**ompiler
- ▶ a parser-generator
- ▶ Input: a Context-Free Grammar ...
- ▶ Output: parser, in file `y.tab.c`
- ▶ Typically used with a lexical analyser, e.g., Lex

Lex

- ▶ **L**exical **A**nalyser
- ▶ Input: regular expression for each token ...
- ▶ Output: lexical analyser, in file `lex.yy.c`

Both are widely available in Unix/Linux

Lex: lexical analysis

filename.l

```
%  
definitions ...  
%%  
    regexps + code, for each token ...  
%%  
C code ...
```



lex

lex.yy.c



```
..... yylex() .....
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Yacc: parser generation

filename.y

```
declarations (incl. token names) ...  
%%  
grammar: production rules ...  
%%  
C code ...
```

yacc

y.tab.c

```
..... yyparse() .....
```

calls yylex() to get tokens

- ▶ Compile `y.tab.c` and `lex.yy.c` using, say, `cc`.
- ▶ yields an executable parser.
- ▶ It can evaluate as it parses.
- ▶ See Assignment 1.

Conflict resolution in Yacc:

- ▶ Shift-reduce: shift
- ▶ Reduce-reduce: use the rule listed first.

Assignment Project Exam Help

- ▶ Construct a parse tree for given string and grammar.
- ▶ Understand how a Shift-reduce Parser works.
- ▶ Start using Lex and Yacc.

<https://powcoder.com>

Add WeChat powcoder