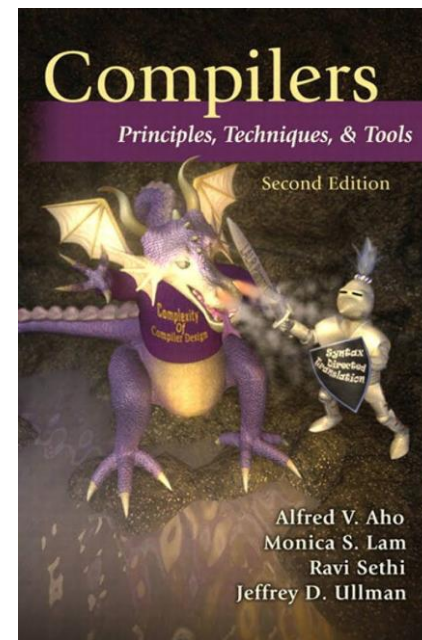


Compiler

Lec 07

Book

Compilers: Principles, Techniques, and Tools is a computer science textbook by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman about compiler construction.



PowerPoint

<http://www.bu.edu.eg/staff/ahmedaboalatah14-courses/14779>

The screenshot displays a web interface for Benha University. At the top, a blue header contains the university logo, the name 'Benha University', and a staff search bar with the text 'Welcome: Ahmed Hassan Ahmed Abu El Atta (Log out)'. Below the header, a navigation menu on the left lists various university-related links. The main content area shows the user's current location: 'You are in: Home/Courses/Compilers'. The course details for 'Compilers' are presented in a table with blue headers and white content cells. The table includes fields for Course name, Level, Last year taught, Course description, Course password, Course files, Course URLs, Course assignments, and Course Exams & Model Answers. Each field has a corresponding 'add' or 'edit' link. A vertical sidebar on the right contains social media icons for Google, Benha University, RG, LinkedIn, Facebook, Twitter, Google+, YouTube, WordPress, and a general social media icon, along with an '(edit)' link at the bottom.

Benha University

Staff Search: Welcome: Ahmed Hassan Ahmed Abu El Atta (Log out)

You are in: [Home/Courses/Compilers](#) [Back To Courses](#)

Ass. Lect. Ahmed Hassan Ahmed Abu El Atta :: Course Details:
Compilers [add course](#) | [edit course](#)

Course name	Compilers
Level	Undergraduate
Last year taught	2018
Course description	Not Uploaded
Course password	
Course files	add files
Course URLs	add URLs
Course assignments	add assignments
Course Exams & Model Answers	add exams

Benha University

Home

النسخة العربية

My C.V.

About

Courses

Publications

Inlinks(Competition)

Theses

Reports

Published books

Workshops / Conferences

Supervised PhD

Supervised MSc

Supervised Projects

Education

Language skills

Academic Positions

Administrative Positions

Google

Benha University

RG

in

f

Twitter

g+

YouTube

W

social media icon

social media icon

social media icon

social media icon

(edit)

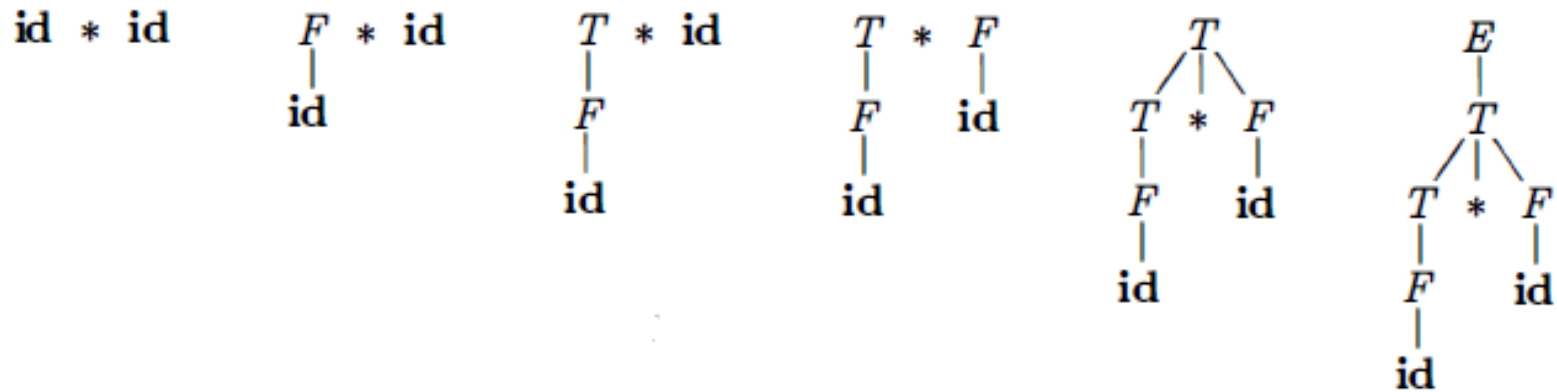
Syntax Analysis

PART IV

Bottom-Up Parsing

A bottom-up parse corresponds to the construction of a parse tree for an input string beginning at the **leaves** (the **bottom**) and working up towards the **root** (the **top**) .

Bottom-Up Parsing (Cont.)



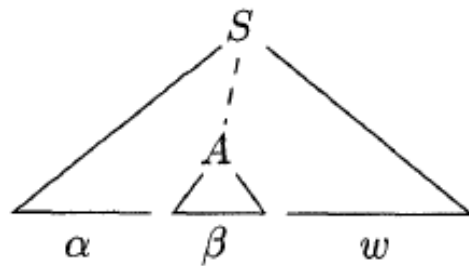
$\text{id} * \text{id} \rightarrow F * \text{id} \rightarrow T * \text{id} \rightarrow T * F \rightarrow T \rightarrow E$

$E \Rightarrow T \Rightarrow T * F \Rightarrow T * \text{id} \Rightarrow F * \text{id} \Rightarrow \text{id} * \text{id}$

Handle Pruning

a "handle" is a substring that **matches the body of a production**, and whose reduction represents one step along the reverse of a rightmost derivation.

RIGHT SENTENTIAL FORM	HANDLE	REDUCING PRODUCTION
$id_1 * id_2$	id_1	$F \rightarrow id$
$F * id_2$	F	$T \rightarrow F$
$T * id_2$	id_2	$F \rightarrow id$
$T * F$	$T * F$	$E \rightarrow T * F$

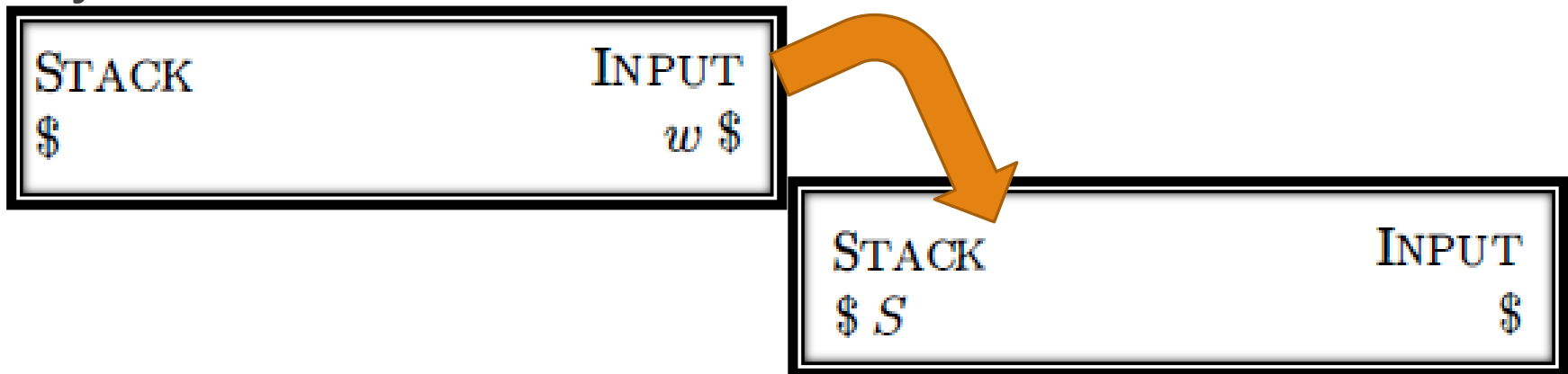


A handle $A \rightarrow \beta$ in the parse tree for $\alpha\beta w$

Shift-Reduce Parsing

Shift-reduce parsing is a form of bottom-up parsing in which a stack holds grammar symbols and an input buffer holds the rest of the string to be parsed.

The **handle** always appears at the **top of the stack** just before it is identified as the handle.



Shift-Reduce Parsing(Cont.)

1. *Shift*. Shift the next input symbol onto the top of the stack.
2. *Reduce*. The right end of the string to be reduced must be at the top of the stack. Locate the left end of the string within the stack and decide with what nonterminal to replace the string.
3. *Accept*. Announce successful completion of parsing.
4. *Error*. Discover a syntax error and call an error recovery routine.

Example

$$\begin{aligned}
 E &\rightarrow E + T \mid T \\
 T &\rightarrow T * F \mid F \\
 F &\rightarrow (E) \mid \mathbf{id}
 \end{aligned}$$

STACK	INPUT	ACTION
\$	$\mathbf{id}_1 * \mathbf{id}_2 \$$	shift
$\$ \mathbf{id}_1$	$* \mathbf{id}_2 \$$	reduce by $F \rightarrow \mathbf{id}$
$\$ F$	$* \mathbf{id}_2 \$$	reduce by $T \rightarrow F$
$\$ T$	$* \mathbf{id}_2 \$$	shift
$\$ T *$	$\mathbf{id}_2 \$$	shift
$\$ T * \mathbf{id}_2$	$\$$	reduce by $F \rightarrow \mathbf{id}$
$\$ T * F$	$\$$	reduce by $T \rightarrow T * F$
$\$ T$	$\$$	reduce by $E \rightarrow T$
$\$ E$	$\$$	accept

LR Parsing: Simple LR

LR(k) parsing;

- the "L" is for left-to-right scanning of the input,
- the "R" for constructing a rightmost derivation in reverse, and
- the k for the number of input symbols of lookahead that are used in making parsing decisions.

The cases $k = 0$ or $k = 1$ are of practical interest, and we shall only consider LR parsers with $k \leq 1$ here.

When (k) is omitted, k is assumed to be 1 .

LR(0) Item

An LR(0) item of G is a **production** of G with the **dot** at some position of the body:

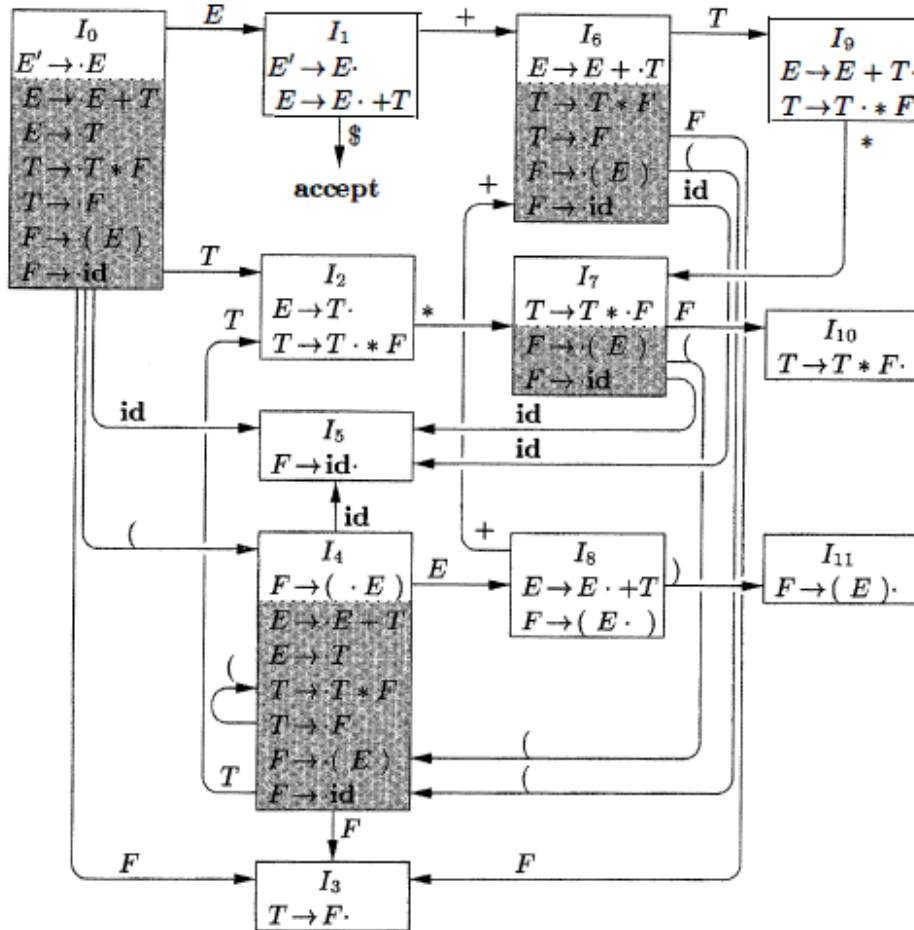
- For $A \rightarrow XYZ$ we have following items
 - $A \rightarrow \cdot XYZ$
 - $A \rightarrow X \cdot YZ$
 - $A \rightarrow XY \cdot Z$
 - $A \rightarrow XYZ \cdot$
- In a state having $A \rightarrow \cdot XYZ$ we hope to see a string derivable from XYZ next on the input.
- The production $A \rightarrow \epsilon$ generates only one item, $A \rightarrow \cdot$.

Closure of Item Sets

If I is a set of items for a grammar G , then $\text{Closure}(I)$ is the set of items constructed from I by the two rules:

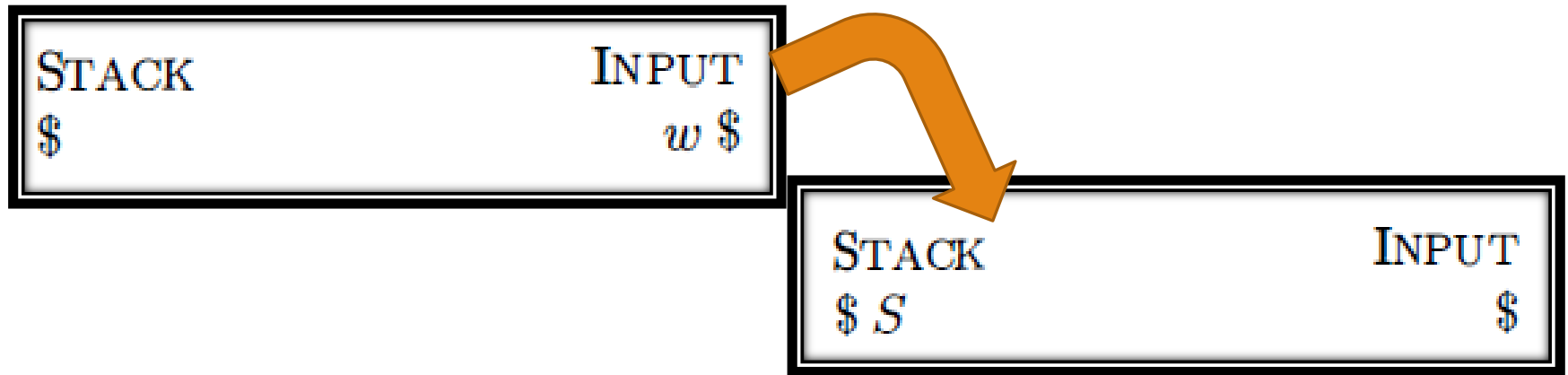
1. Initially, add every item in I to $\text{CLOSURE}(I)$.
2. If $A \rightarrow \alpha \cdot B \beta$ is in $\text{CLOSURE}(I)$ and $B \rightarrow \gamma$ is a production, then add the item $B \rightarrow \cdot \gamma$ to $\text{CLOSURE}(I)$, if it is not already there. Apply this rule until no more new items can be added to $\text{CLOSURE}(I)$.

LR(0) Automaton



$E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid id$

$S' \rightarrow S.$



To construct the canonical LR(0) collection for a grammar, we define an **augmented grammar**

Augmented Grammar

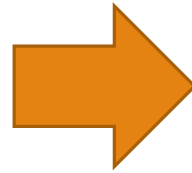
If G is a grammar with start symbol S , then G' , the augmented grammar for G , is G with a new start symbol S' and production $S' \rightarrow S$

The purpose of this new starting production is to indicate to the parser when it should stop parsing and announce acceptance of the input.

That is, acceptance occurs when and only when the parser is about to reduce by $S' \rightarrow S$.

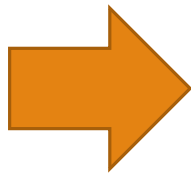
Example

$E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid \text{id}$



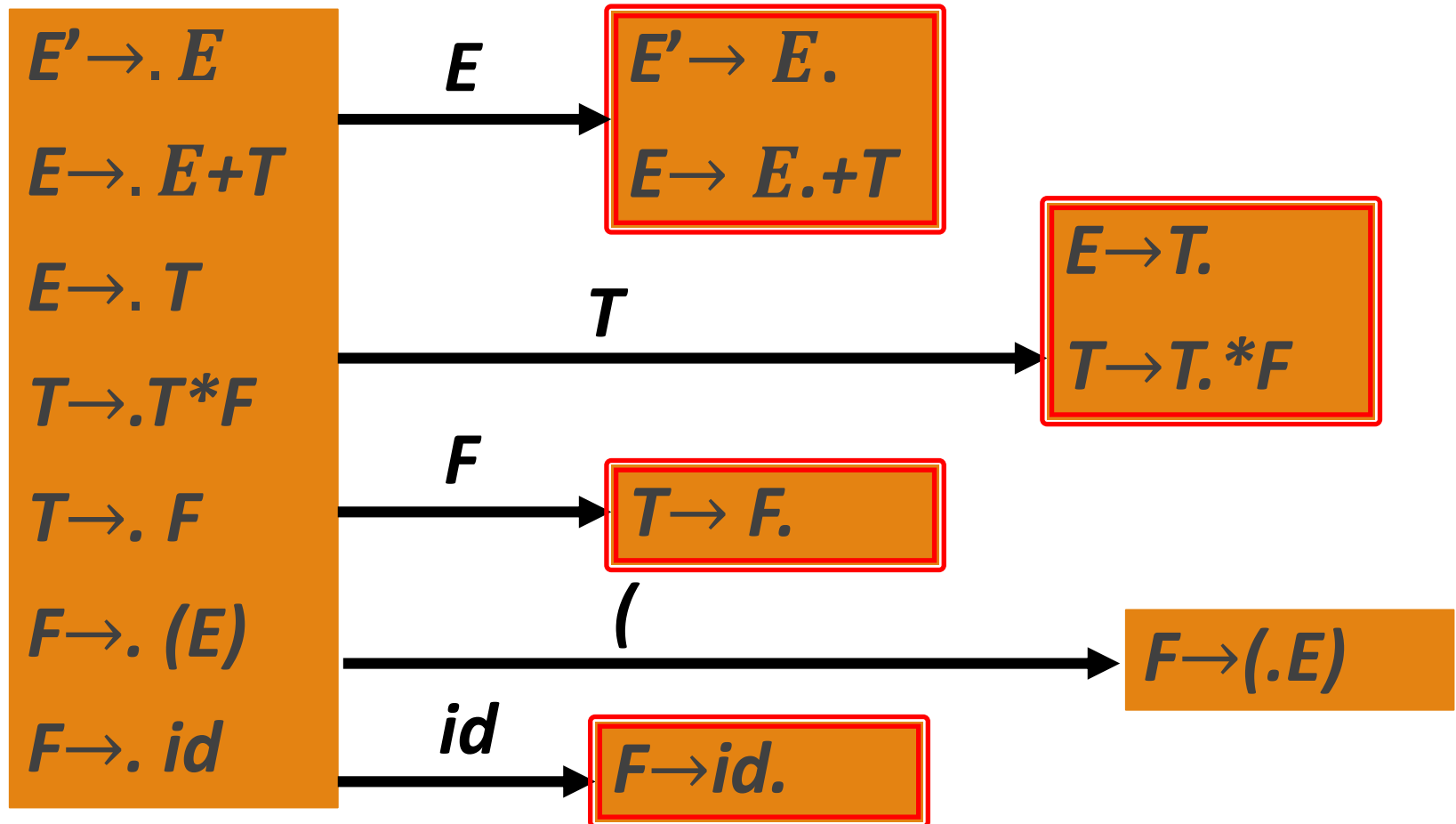
$E' \rightarrow E$
 $E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $E \rightarrow (E) \mid \text{id}$

$E' \rightarrow . E$



$E' \rightarrow . E$
 $E \rightarrow . E + T$
 $E \rightarrow . T$
 $T \rightarrow . T * F$
 $T \rightarrow . F$
 $F \rightarrow . (E)$
 $F \rightarrow . \text{id}$

Example



LR(0) Example

$S \rightarrow aSb \mid aSc \mid db$

Add $S' \rightarrow S$

Then start with $S' \rightarrow .S$

