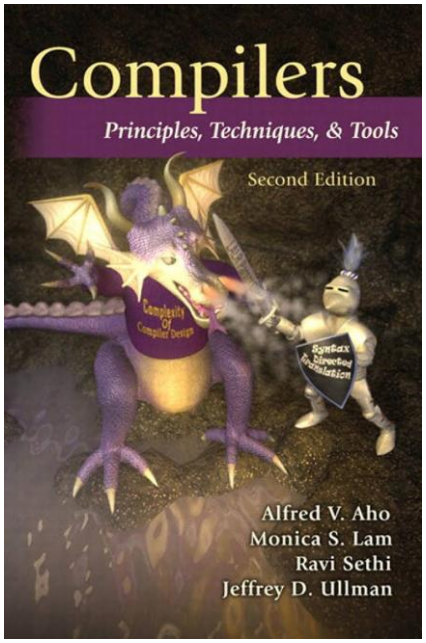
Compiler

Lec 07

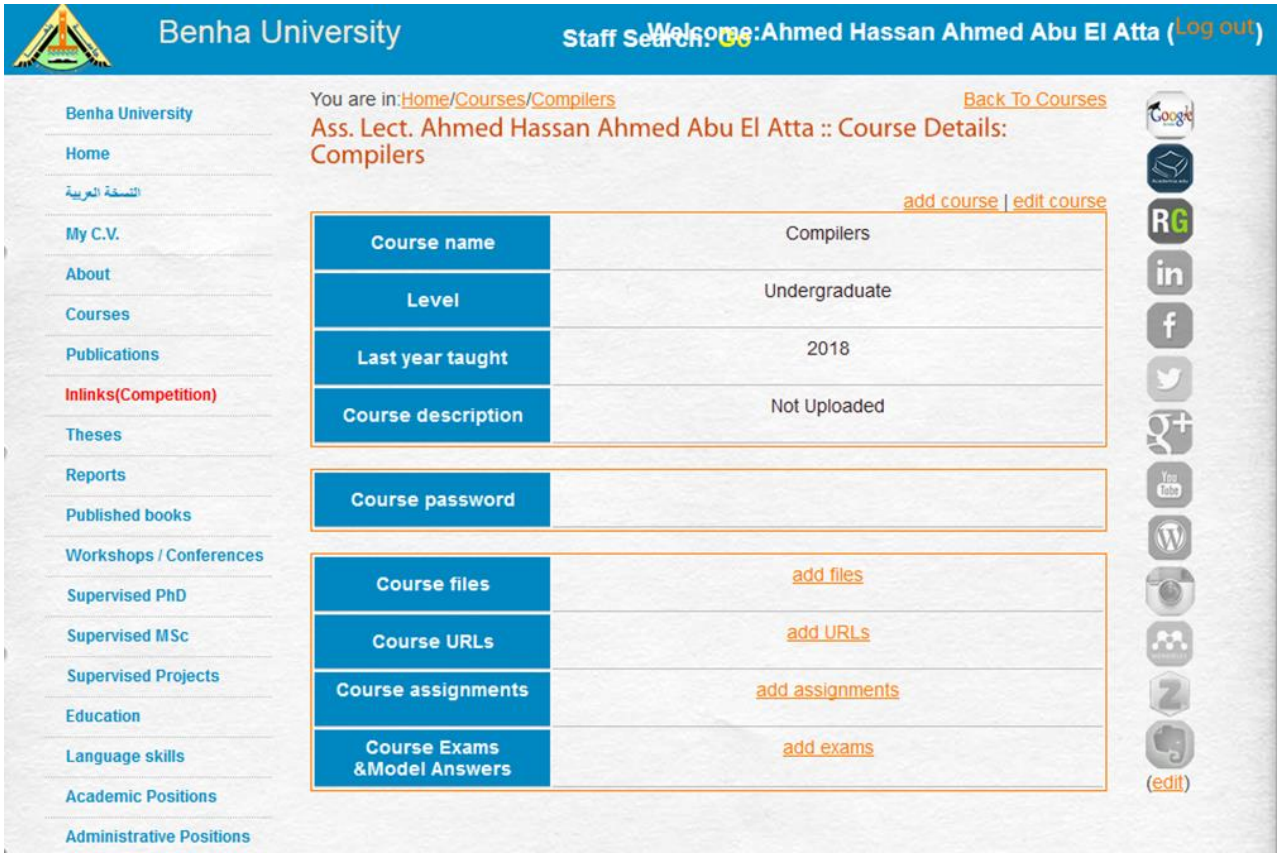
Book

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



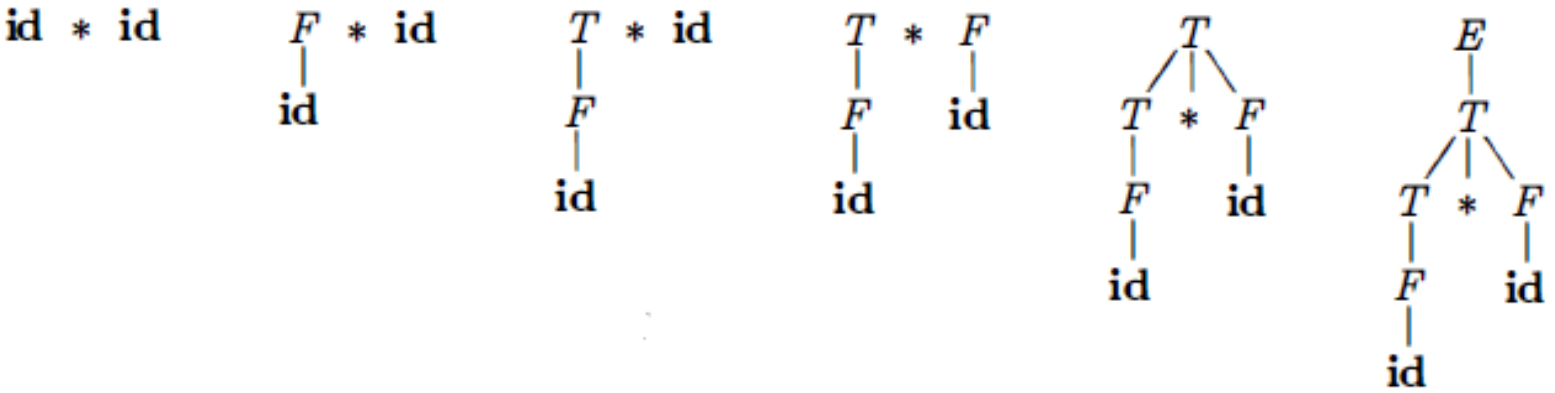
Syntax Analysis

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.)



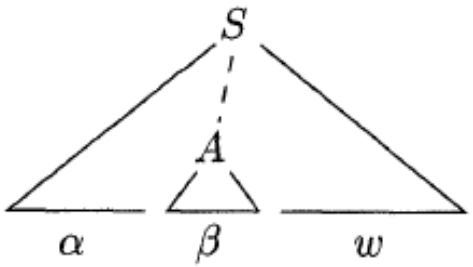
id * id→ F * id → T * id → T * F → T → E

E => T => T * F => T * id => F * id => id * 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
$\mathbf{id}_1 * \mathbf{id}_2$	$\mathbf{id}_1$	$F \to \mathbf{id}$
$F * \mathbf{id}_2$	F	$T \to F$
$T * \mathbf{id}_2$	$\mathbf{id}_2$	$F \to \mathbf{id}$
$T * F$	$T * F$	$E \to T * F$

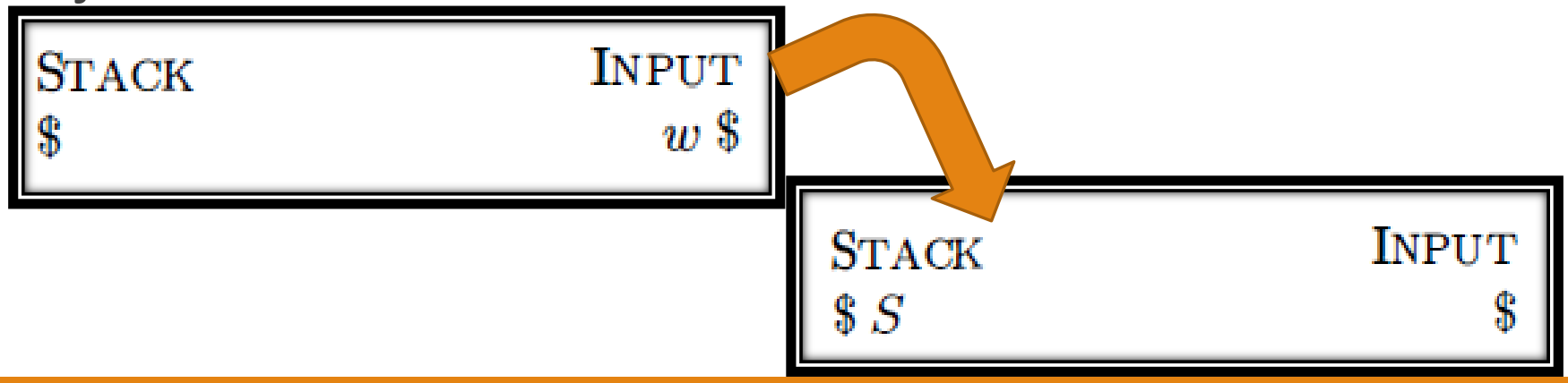


A handle $A \to \beta$ in the parse tree for $\alpha\beta w$

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

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

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:

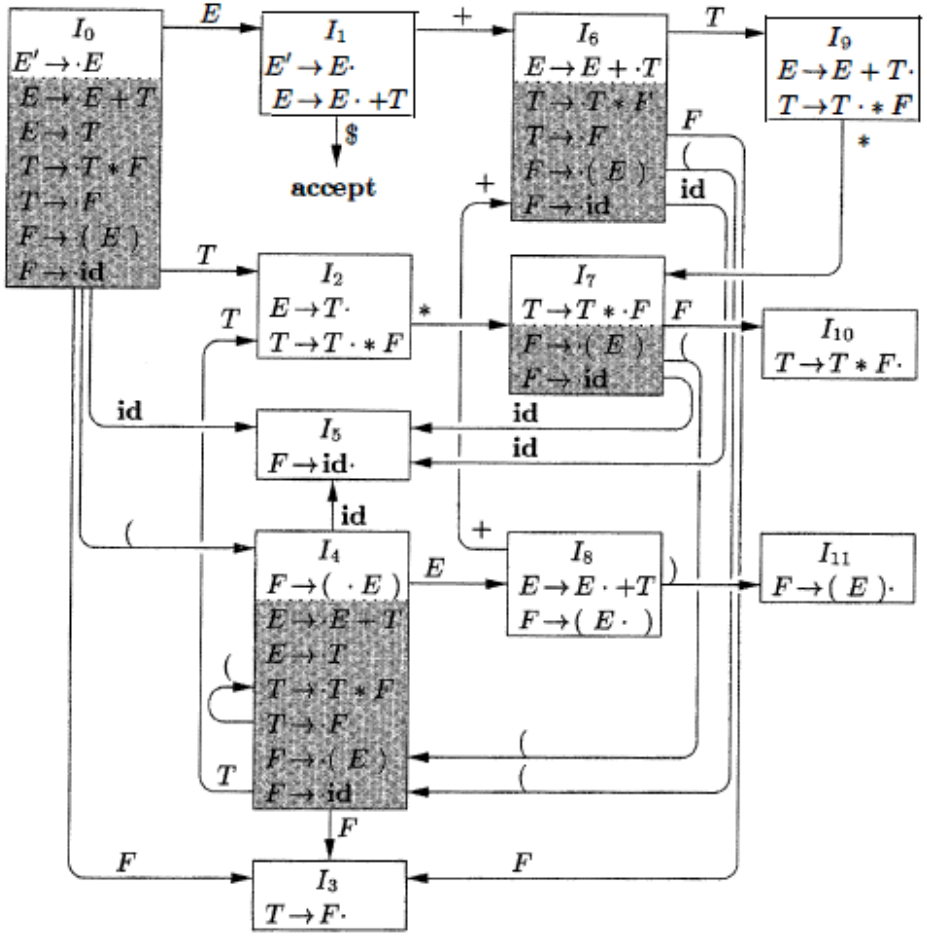
- For A->XYZ we have following items
  - **A->.XYZ**
  - **A->X.YZ**
  - **A->XY.Z**
  - **A->XYZ.**
- In a state having A->.XYZ we hope to see a string derivable from XYZ next on the input.
- The production A->ϵ generates only one item, A -> .

Closure of Item Sets

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

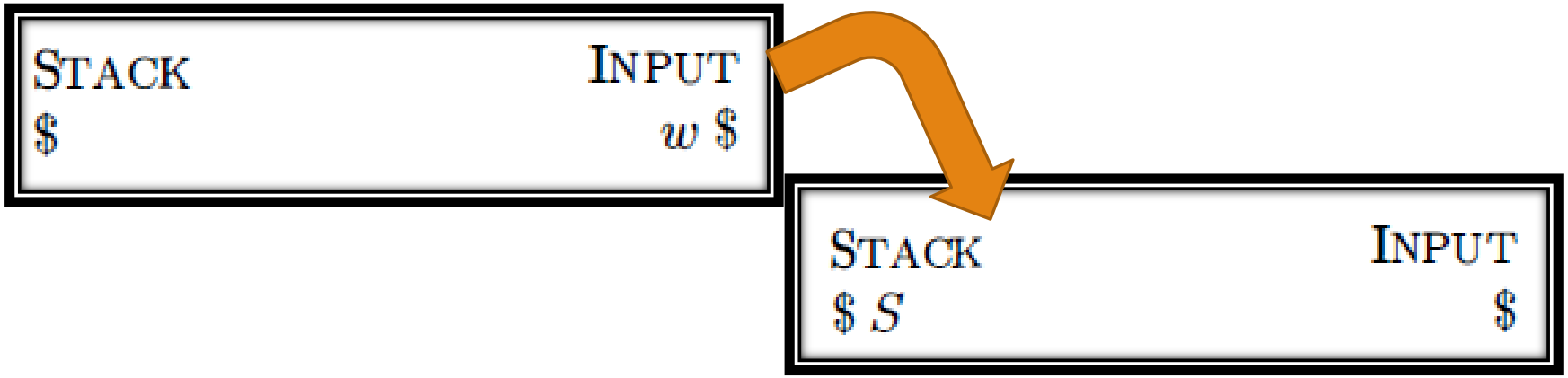
1. Initially, add every item in I to CLOSURE(I).
2. If $A \to \alpha \cdot B\beta$ is in CLOSURE(I) and $B \to \gamma$ is a production, then add the item $B \to \cdot\gamma$ to CLOSURE(I), if it is not already there. Apply this rule until no more new items can be added to CLOSURE(I).

LR(0) Automaton



$$
\begin{array}{lcl}
E & \to & E + T \mid T \\
T & \to & T * F \mid F \\
F & \to & (E) \mid \mathbf{id}
\end{array}
$$

$S' \rightarrow S.$



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

Augmented Grammar

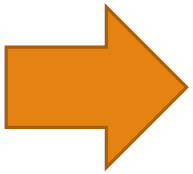
If G is a grammar with start symbol S, then G' , the augmented grammar for G, is G with a new start symbol S' and production S'→S

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'→S.

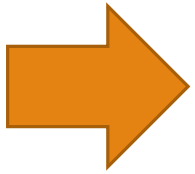
Example

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



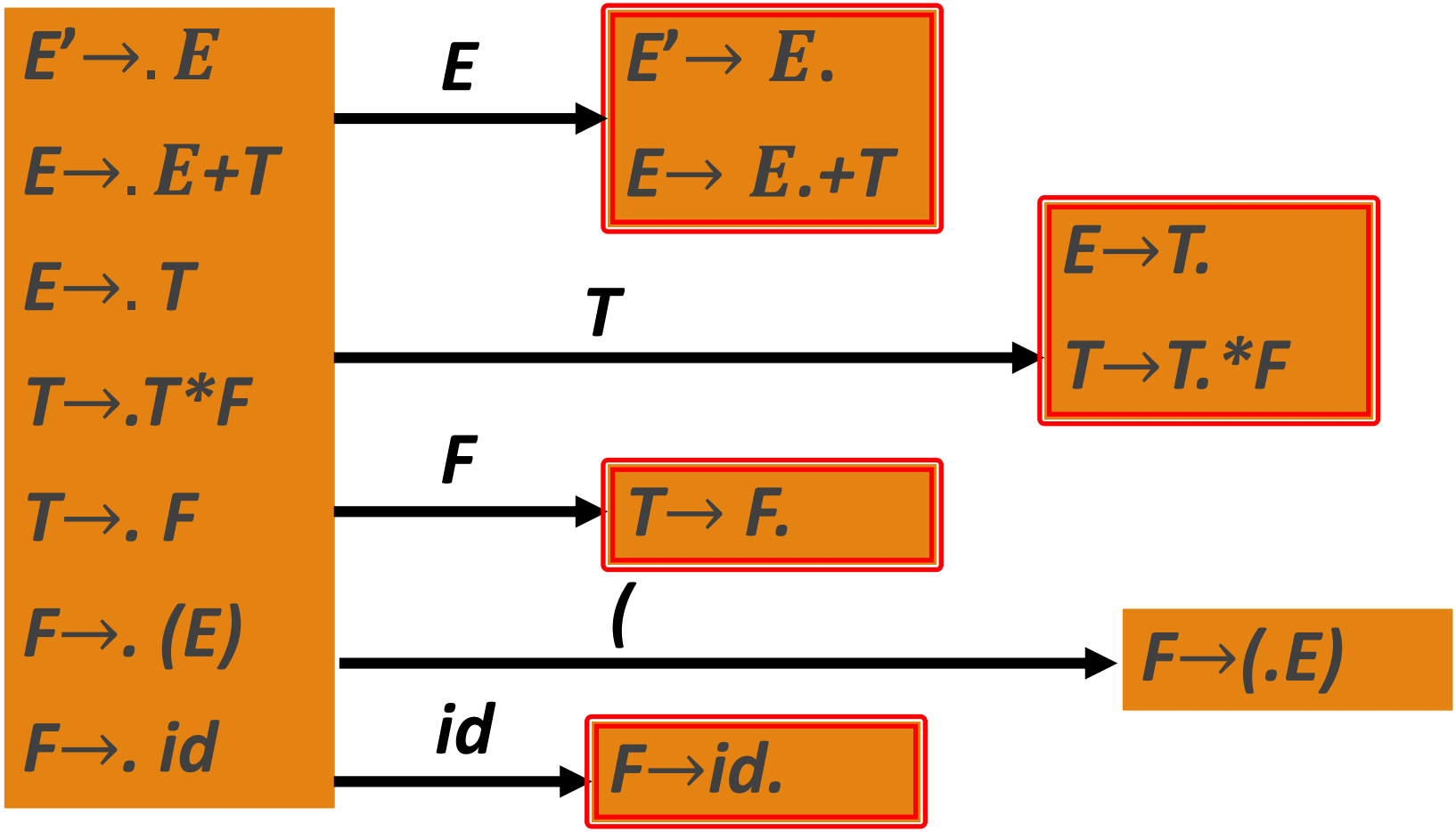
$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
E & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

E'→. E



E'→. E
E→. E+T
E→. T
T→.T*F
T→. F
F→. (E)
F→. id

Example



LR(0) Example

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

Add $S' \rightarrow S$

Then start with $S' \rightarrow .S$