# Analyse Syntaxique ascendante

# LR Parsing Techniques

# Plan

- Introduction
- Analyseur à décalage-réduction  (Shift-Reduce Parsers)
- LR Parsers
  - LR(1) Parsing
  - SLR(1)Parsing
  - LALR(1

# Introduction(1)

- **Analyseurs descendants**
  - **commence la construction de l'arbre d'analyse à partir du la racine de l'arbre et se déplacer vers le bas (vers les feuilles).**
  - **Facile à mettre en œuvre à la main, mais travailler avec les grammaires restreintes.**
  - **exemple: analyseur prédictifs LL(1)**

# Introduction(2)

- **Analyseurs ascendants (Bottom-up)**
  - **Cherchent à construire un arbre de dérivation droite à partir des feuilles selon un parcours d'un parcours en profondeur d'abord.**
  - **Les étapes de réduction tracent une dérivation à droite à l'envers.**
  - **Suitable pour un générateur d'analyseur syntaxique automatique**
  - **Peut gérer une grande classe de grammaires.**
  - **exemples: shift-reduce parser (ou LR (k) parsers)**

**Grammaire**

$S \rightarrow aABe$

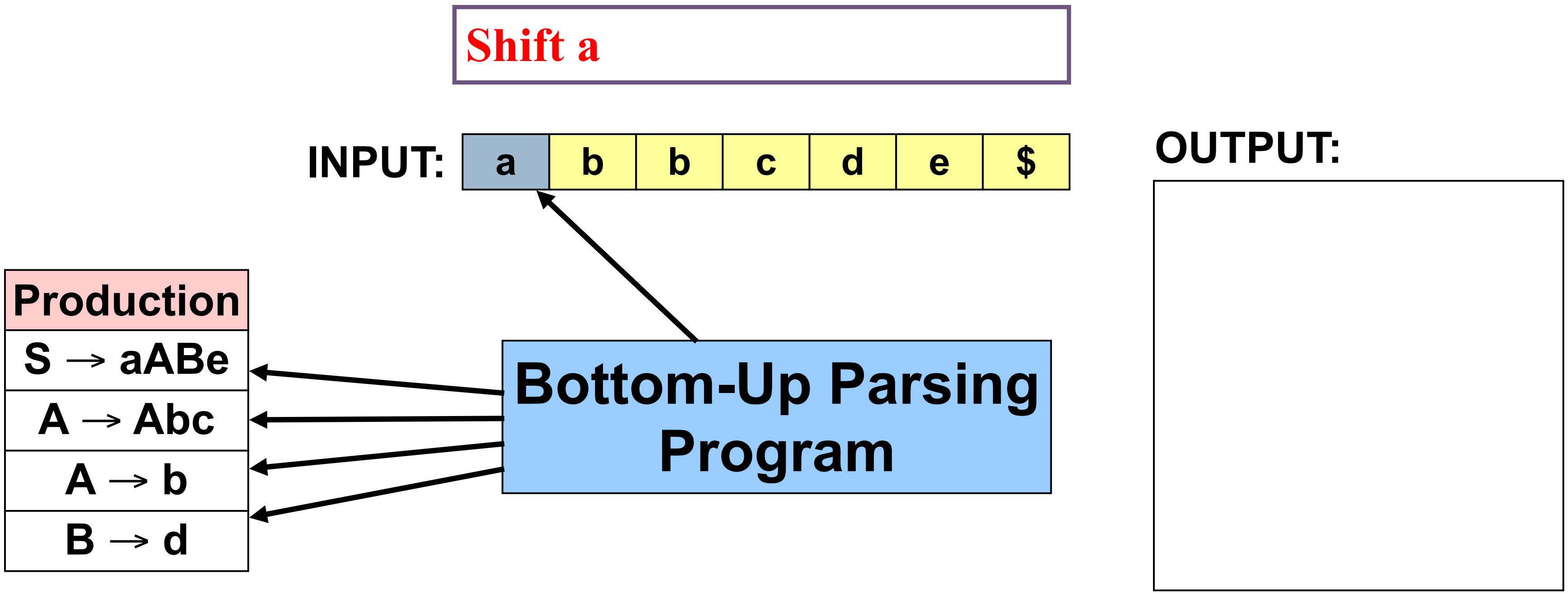
$A \rightarrow Abc \mid b$

$B \rightarrow d$

**input string : abbcde.**

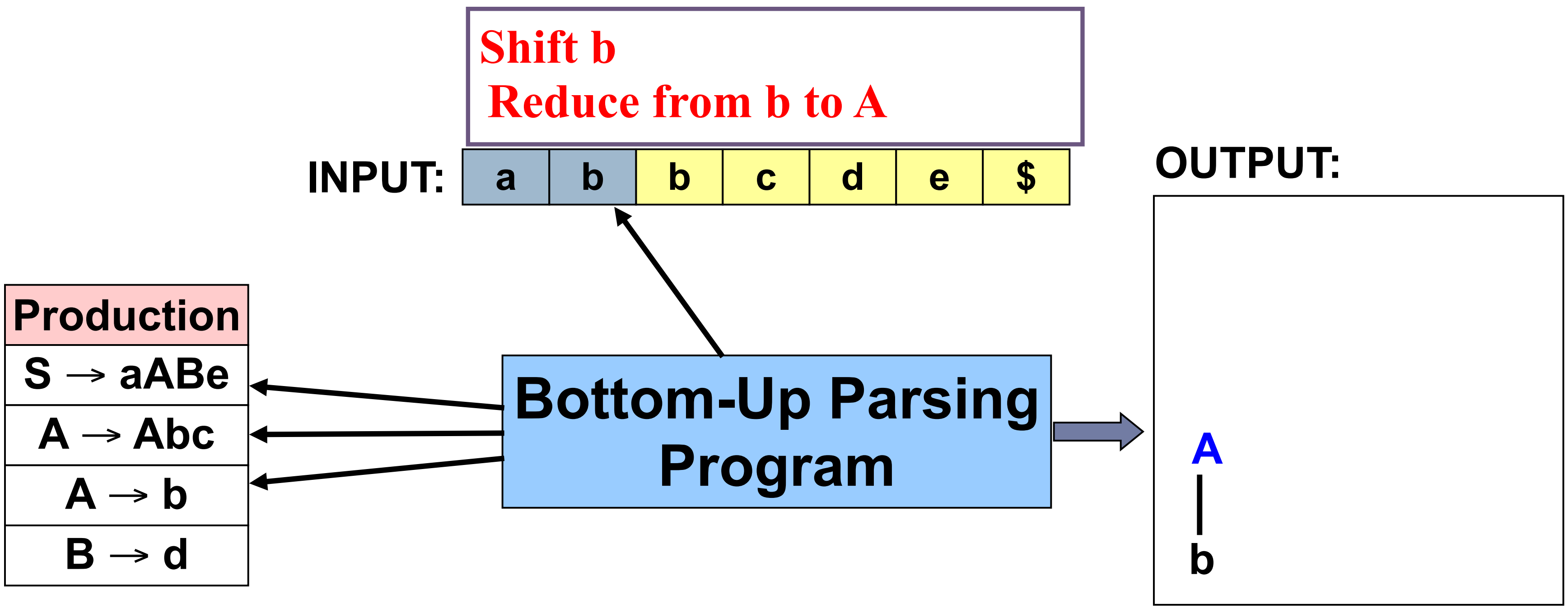
# Introduction(3)

## Exemple d'analyseur Bottom-Up



# Introduction(4)

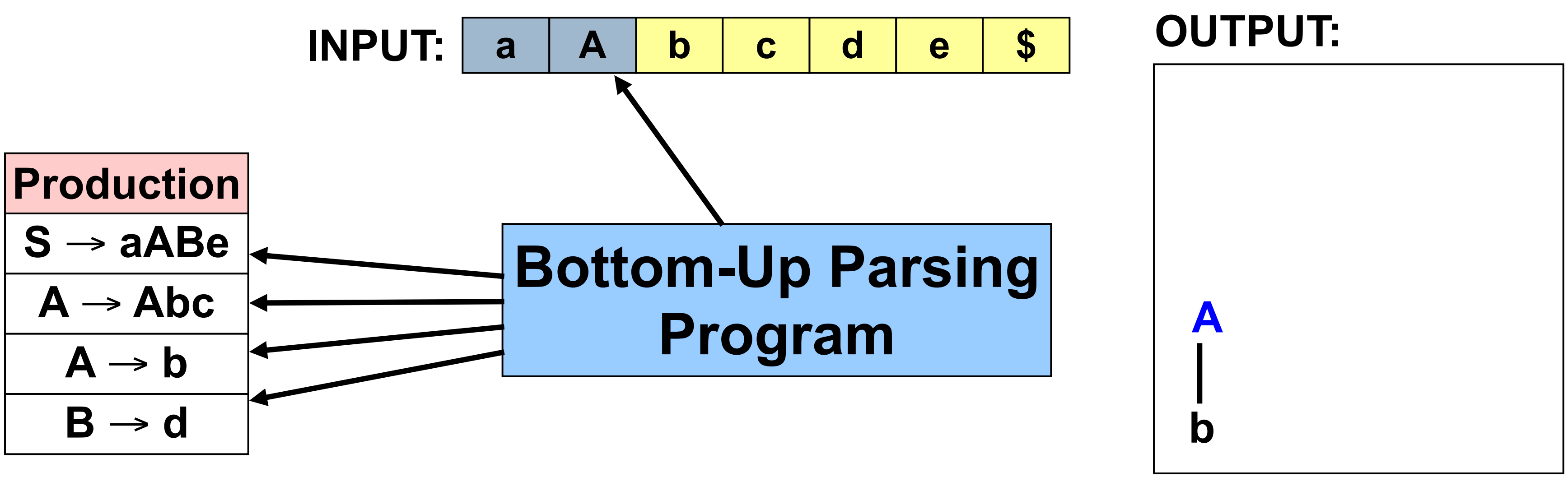
## Exemple d’analyseur Bottom-Up



# Introduction(5)

## Exemple d’analyseur Bottom-Up

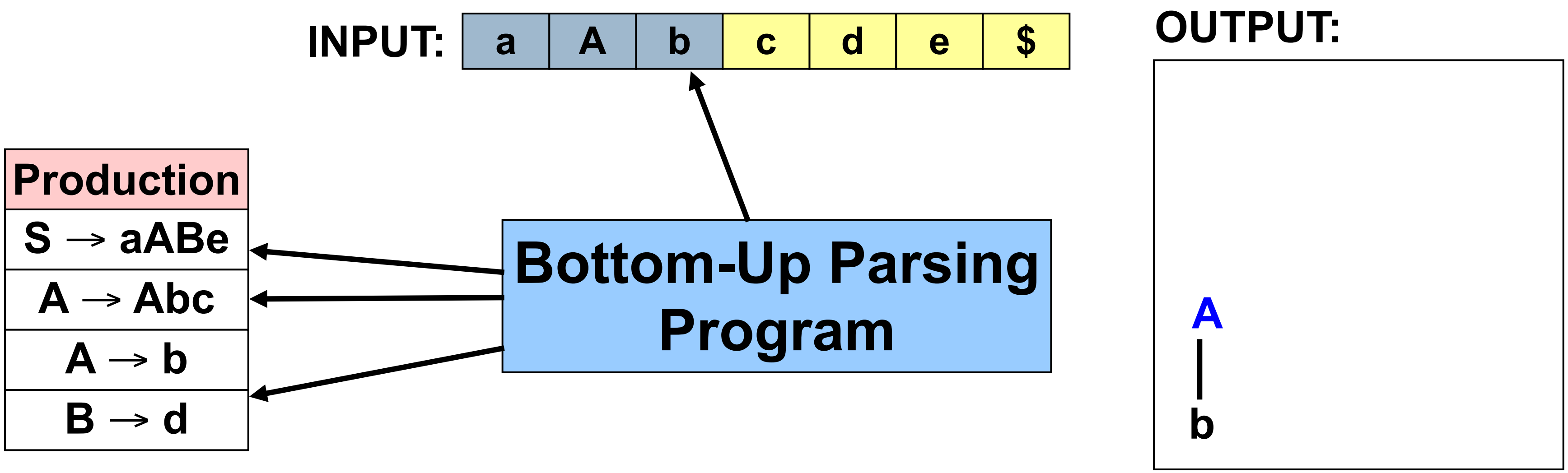
Shift A



# Introduction(6)

## Exemple d’analyseur Bottom-Up

Shift b



# Introduction(7)

## Exemple d’analyseur Bottom-Up

Shift c
Reduce from Abc to A

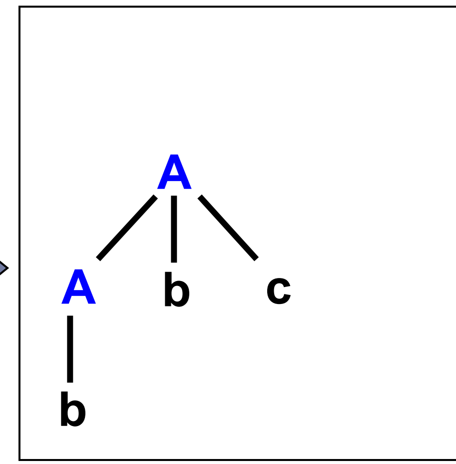
INPUT:

| | | | | | | |
|---|---|---|---|---|---|---|
| a | A | b | c | d | e | $ |

| Production |
|---|
| S → aABe |
| A → Abc |
| A → b |
| B → d |

Bottom-Up Parsing Program

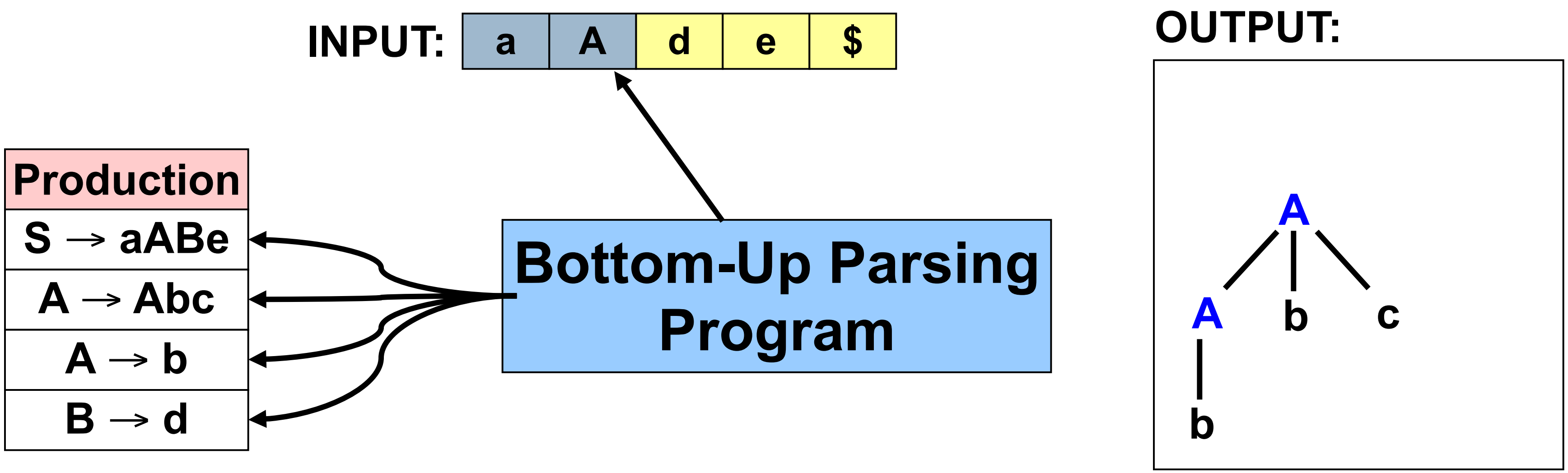
OUTPUT:



# Introduction(8)

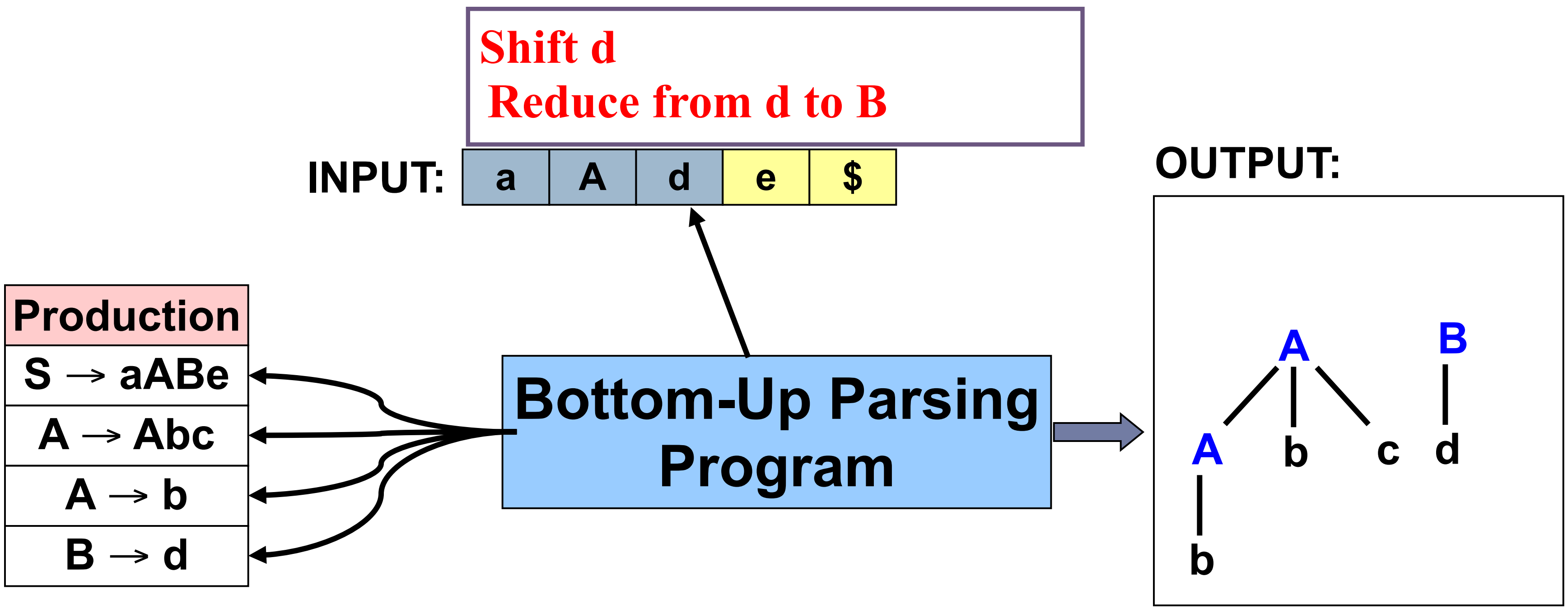
## Exemple d'analyseur Bottom-Up

Shift A



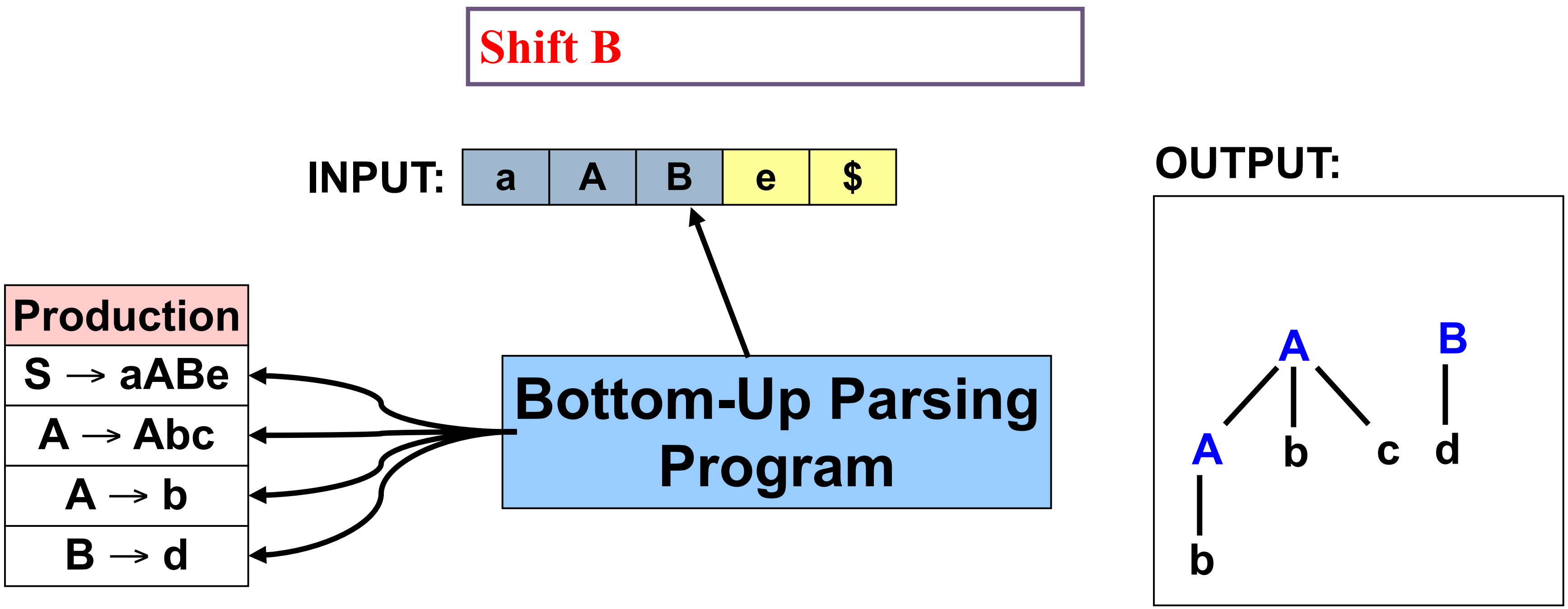
# Introduction(9)

## Exemple d'analyseur Bottom-Up



# Introduction(10)

## Bottom-Up Parser Example



# Introduction(11)

## Exemple d’analyseur Bottom-Up

Shift e
Reduce from aABe to S

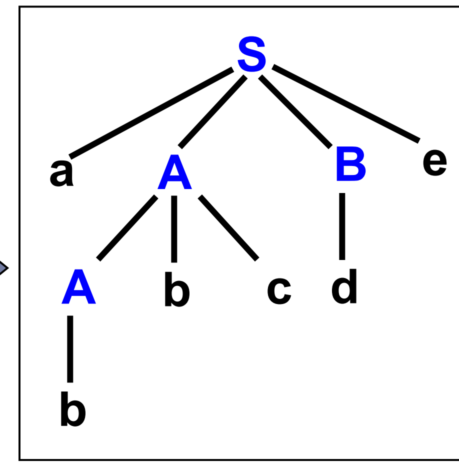
INPUT:

| a | A | B | e | $ |
|---|---|---|---|---|

| Production |
|---|
| S → aABe |
| A → Abc |
| A → b |
| B → d |

Bottom-Up Parsing Program

OUTPUT:



# Introduction(12)

## Exemple d’analyseur Bottom-Up

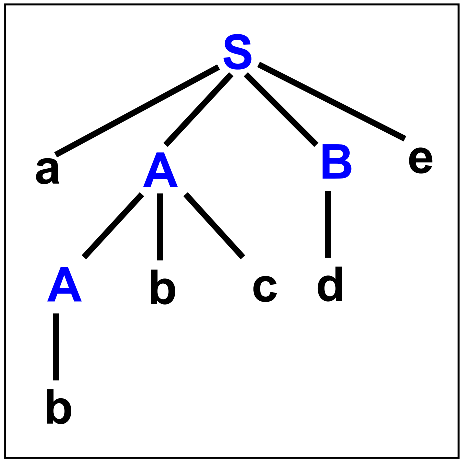
Shift S
Hit the target $

INPUT: S | $

Bottom-Up Parsing Program

| Production |
|---|
| S → aABe |
| A → Abc |
| A → b |
| B → d |

OUTPUT:




**Cet analyseur est connu comme un analyseur LR parce il parcourt l'entrée de gauche à droite, et il construit une dérivation plus à droite dans l'ordre inverse.**

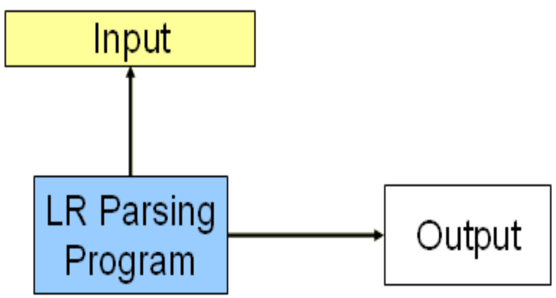
# Introduction(13)

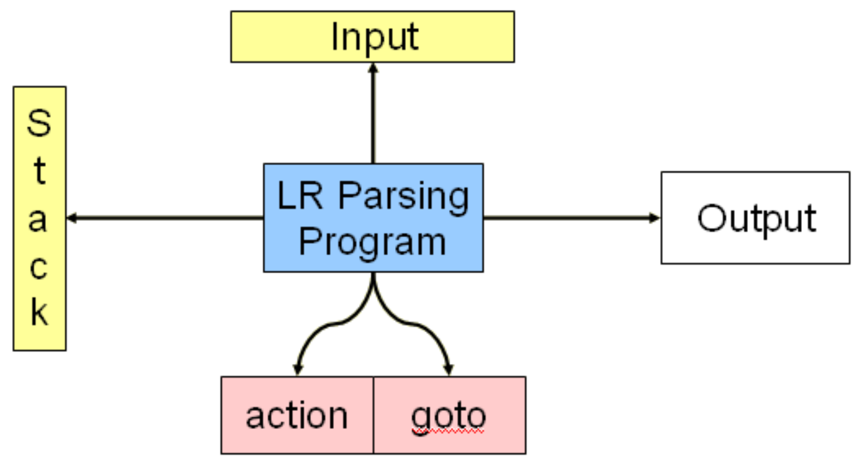
- **Conclusion**
  - Le balayage des productions pour faire correspondre avec **des poignées** dans la chaîne d'entrée
  - Le Backtracking rend la procédure utilisé dans l'exemple précédent très inefficace.

*Pouvons-nous faire mieux? Discuter plus tard !!!*

*Architecture précédente*

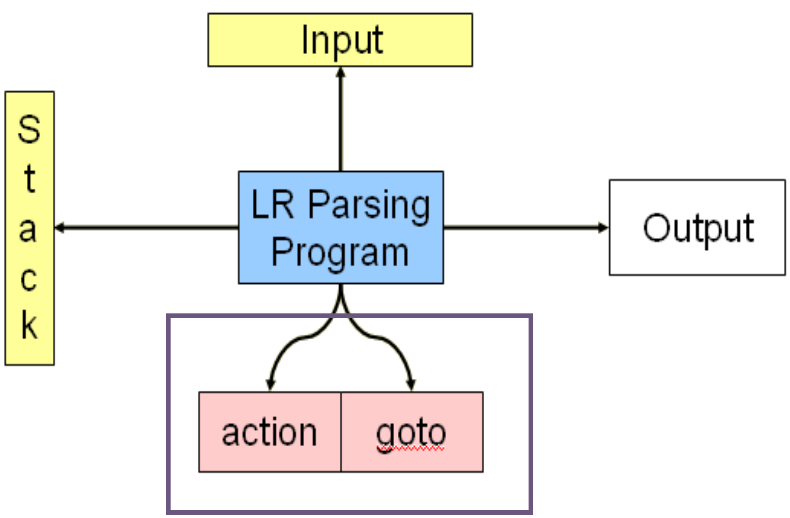
*Renouveler l'Architecture*



# Shift-Reduce Parsers(5)

- **Les analyseurs LR sont entraînés par deux tableaux:**
  - **Table d'action, qui précise quel action à prendre : Shift, reduction, accepter ou d'une erreur**
  - **Table Goto, qui spécifie les transitions d'états pour signaler le passage de la machine à états finis.**
    - **Nous empilons des Etats, plutôt que des symboles sur la pile**
- **Chaque état représente les éventuelles sous-arbres de l'arbre d'analyse**



# Shift-Reduce Parsers(6)

- grammar $G_0$

1. <program>→ **begin** <stmts> **end $**
2. <stmts> → **SimpleStmt ;** <stmts>
3. <stmts> → **begin** <stmts> **end ;** <stmts>
4. <stmts> → **ϵ**

**Action Table**

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | S | S | | | S | | S | | | S | | |
| **end** | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | S | | | S | | S | | | S | | |

**Figure 6.2** A Shift-Reduce **action** Table for $G_0$

Case blanche

—> ERREUR

**Goto Table**

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce **go_to** Table for $G_0$

# Shift-Reduce Parsers(7)

```
void shift_reduce_driver(void)
{
    /* Push the Start State, S0,
     * onto an empty parse stack. */
    push(S0);
    while (TRUE) {                /* forever */
    /* Let S be the top parse stack state;
     * let T be the current input token.*/
        switch (action[S][T]) {
        case ERROR:
        announce_syntax_error();
                break;
        case ACCEPT:
        /* The input has been correctly
         * parsed. */
                clean_up_and_finish();
                return;
        case SHIFT:
            push(go_to[S][T]);
            scanner(&T);
                /* Get next token. */
            break;
        case REDUCEi:
        /* Assume i-th production is
         * X → Y1 · · · Ym.
         * Remove states corresponding to
         * the RHS of the production. */
            pop(m);
            /* S' is the new stack top. */
            push(go_to[S'][X]);
            break;
        }
    }
}
```

# Shift-Reduce Parsers(8)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ϵ

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| end | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (1) | 0 | begin SimpleStmt ; SimpleStmt ; end $ | Shift 1 |

action table

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(9)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ε
- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (2) | 0,1 | SimpleStmt ; SimpleStmt ; end $ | Shift 5 |

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| end | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

Figure 6.3 A Shift-Reduce go_to Table for $G_0$

action table

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(10)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ϵ

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| end | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | . | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (3) | 0,1,5 | ; SimpleStmt ; end $ | Shift 6 |

action table

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(10)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ε

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | $\boxed{5}$ | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (4) | 0,1,5,6 | SimpleStmt ; end $ | Shift 5 |

action table

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| <u>begin</u> | S | S | | | S | | S | | | S | | |
| <u>end</u> | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(10)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ε
- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (5) | 0,1,5,6,5 | ; end $ | Shift 6 |

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| end | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

action table

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(13)

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- **grammar** $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ϵ
- **tracing steps**

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (6) | 0,1,5,6,5,6 | end $ /* goto(6,<stmts>) = 10 */ | Reduce 4 |

**action table**

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(13)

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ε
- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (6) | 0,1,5,6,5,6 | end $ /* goto(6,<stmts>) = 10 */ | Reduce 4 |

```
case REDUCE_i:
/* Assume i-th production is
 * X → Y_1 · · · Y_m.
 * Remove states corresponding to
 * the RHS of the production. */
  pop(m);
  /* S' is the new stack top. */
  push(go_to[S'][X]);
```

**action table**

| Symbol | Stat | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | | | | |
| begin | S | S | | | S | | S | | | | | |
| end | | R4 | S | | R4 | | R | | | | | |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(14)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ε
- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (6) | 0,1,5,6,5,6,10 | end $ /* goto(6,<stmts>) = 10 */ | Reduce 4 |

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| end | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

action table

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(14)

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ε
- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (8) | 0,1 | end $ /* goto(1,<stmts>) = 2 */ | Reduce 2 |

**action table**

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(14)

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- grammar $G_0$
  1. <program> → begin<stmts>end$
  2. <stmts> → SimpleStmt;<stmts>
  3. <stmts> → begin<stmts>end;<stmts>
  4. <stmts> → ε
- tracing steps

| Step | Parse Stack | Remaining Input | | Action |
|---|---|---|---|---|
| (8) | 0,1 | end $ | /* goto(1,<stmts>) = 2 */ | Reduce 2 |

**action table**

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(14)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ϵ
- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (9) | 0,1,2 | end $ | Shift 3 |

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

**action table**

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(14)

- grammar $G_0$
  1. <program>→begin<stmts>end$
  2. <stmts> →SimpleStmt;<stmts>
  3. <stmts> →begin<stmts>end;<stmts>
  4. <stmts> →ε

| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **begin** | 1 | 4 | | | 4 | | 4 | | | 4 | | |
| **end** | | | 3 | | | | | 8 | | | | |
| ; | | | | | | 6 | | | 9 | | | |
| SimpleStmt | | 5 | | | 5 | | 5 | | | 6 | | |
| $ | | | | | | | | | | | | |
| <program> | | | | | | | | | | | | |
| <stmts> | | 2 | | | 7 | | 10 | | | 11 | | |

**Figure 6.3** A Shift-Reduce `go_to` Table for $G_0$

- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (10) | 0,1,2,3 | $ | Accept |

**action table**

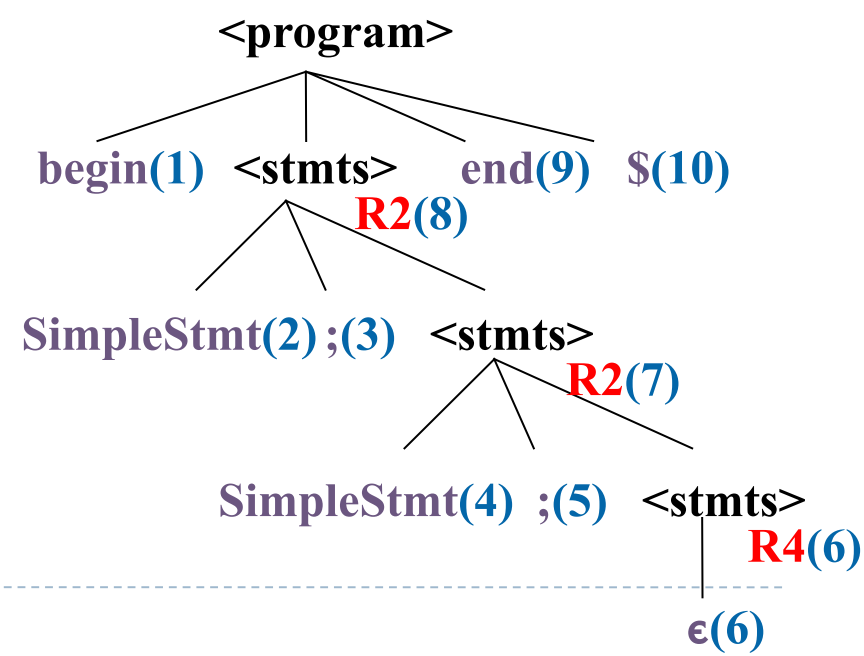
| Symbol | State | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| begin | S | S | | | S | | S | | | S | | |
| end | | R4 | S | | R4 | | R4 | S | | R4 | R2 | R3 |
| ; | | | | | | S | | | S | | | |
| SimpleStmt | | S | | | S | | S | | | S | | |
| $ | | | | A | | | | | | | | |

# Shift-Reduce Parsers(18)

- tracing steps

| Step | Parse Stack | Remaining Input | Action |
|---|---|---|---|
| (1) | 0 | begin SimpleStmt ; SimpleStmt ; end $ | Shift 1 |
| (2) | 0,1 | SimpleStmt ; SimpleStmt ; end $ | Shift 5 |
| (3) | 0,1,5 | ; SimpleStmt ; end $ | Shift 6 |
| (4) | 0,1,5,6 | SimpleStmt ; end $ | Shift 5 |
| (5) | 0,1,5,6,5 | ; end $ | Shift 6 |
| (6) | 0,1,5,6,5,6 | end $ /* goto(6,<stmts>) = 10 */ | Reduce 4 |
| (7) | 0,1,5,6,*5,6,10* | end $ /* goto(6,<stmts>) = 10 */ | Reduce 2 |
| (8) | 0,1,*5,6,10* | end $ /* goto(1,<stmts>) = 2 */ | Reduce 2 |
| (9) | 0,1,2 | end $ | Shift 3 |
| (10) | 0,1,2,3 | $ | Accept |

- grammar $G_0$
  1. <program> → begin <stmts> end $
  2. <stmts> → SimpleStmt ; <stmts>
  3. <stmts> → begin <stmts> end ; <stmts>
  4. <stmts> → ϵ



# Plan

- Introduction
- Analyseur à décalage-réduction
  (Shift-Reduce Parsers)
- LR Parsers
- LR(1) Parsing
- SLR(1)Parsing
- LALR(1

# Les analyseurs LR

- LR (n) n = 0 ~ k
  - Lecture de gauche, dérivation la plus à droite, n anticipation
- Les analyseurs LR sont déterministes
- Aucune action d'analyse de sauvegarde ou de nouvelle tentative
- LR (0):
    - Sans prédiction, lecture à gauche, dérivation la plus à droite, et 0 anticipation
- LR (1):
  - Anticipation à 1 jeton
- En général unAnalyseurs LR (k)
  - Décide la prochaine action en examinant les jetons déjà déplacés plus k jetons d'anticipation
  - Le plus puissant des analyseurs déterministes
  - Difficile à mettre en œuvre

# Construction de la Table LR(0)(1)

- Une règle de production à la forme
  - $A \rightarrow X_1 X_2 \dots X_j$
- En ajoutant un point •, on obtient une configuration (ou item)
  - $A \rightarrow \bullet X_1 X_2 \dots X_j$
  - $A \rightarrow X_1 X_2 \dots X_i \bullet X_{i+1} \dots X_j$
  - $A \rightarrow X_1 X_2 \dots X_j \bullet$
  - Le point • indique de combien la partie droite (RHS) à était empilée.
- un item avec • à la fin RHS $A \rightarrow X_1 X_2 \dots X_j \bullet$ indique que la partie droite RHS doit être réduit en LHS
- un item avec • au début de RHS $A \rightarrow \bullet X_1 X_2 \dots X_j$
  - prédit que RHS va être décaler dans la pile

# Construction de la Table LR(0)(2)

- **un état LR(0) est un ensemble de configurations**
  - c'à'd l'état actuelle de l'analyseur LR(0) est déterminé par un de ses items.
- **l'operation closure0:**
  - si il ya une configuration $B \rightarrow \delta \cdot A\, \rho$ dans l'ensemble alors tous ces configurations de la forme $A \rightarrow \cdot\, \gamma$ sont aussi dans l'ensemble.
- **la configuration initiale est s0 = closure0($\{S \rightarrow \cdot\, \alpha\ \$\}$)**

EX: for grammar $G_1$ :

1. $S' \rightarrow S\$$
2. $S \rightarrow ID | \epsilon$

closure0( { $S \rightarrow \cdot\, S\ \$$ } ) =
{ $S' \rightarrow \cdot\, S\$$,
$S \rightarrow \cdot\, ID$,
$S \rightarrow \epsilon\ \cdot$ }

special case: ϵ

```
Configuration_set closure (configuration_set s)
{
  configuration_set s' = s ;
  do {
     if( B →δ • A ρ ∈ s' for A ∈ V_n ) {
       /* Predict productions with A as the
          left-hand side
        */
          Add all configurations of the form
          A → • γ to s'
      }
   } while (more new configurations can be added) ;
  return 0;
}
```

# Construction de la Table LR(0)(3)

- **Q1: Pourquoi la grammaire utilise $S' \rightarrow S\$$ ?**
  - R: facile de vérifier la fin de l'analyse!

EX: Si S' n'exist pas~

$S \rightarrow ID\$$

$S \rightarrow \epsilon\$$

Quand l'analyse button-up veut réduire au symbole original S, il ya 2 chemin pour le faire! problème! .

EX: for grammar $G_1$ :

1. $S' \rightarrow S\$$
2. $S \rightarrow ID \mid \epsilon$

closure0( { $S' \rightarrow \bullet\, S\ \$$ } ) =

{ $S' \rightarrow \bullet\, S\$$,

$S \rightarrow \bullet\, ID$,

$S \rightarrow \epsilon\, \bullet$ }

# Construction de la Table LR(0)(4)

- Etant donnée une *configuration* s, on peut calculer son successeur, s' , sous un symbole X
  - Dénoter par go_to0(s,X)=s'

```
Configuration_set goto (configuration_set s , symbol x)
{
  Sb = Ø ;
  for (each configuration c ∈ s)
    if( each configuration c ∈ s)
      Add A → βx • γ to sb ;
  /*
   *  That is, we advance the • past the symbol X,
   *  if possible. Configurations not having a
   *  dot preceding an X are not included in sb .
   */

   /* Add new predictions to  sb via closure0. */
  return closure0(sb) ;
}
```

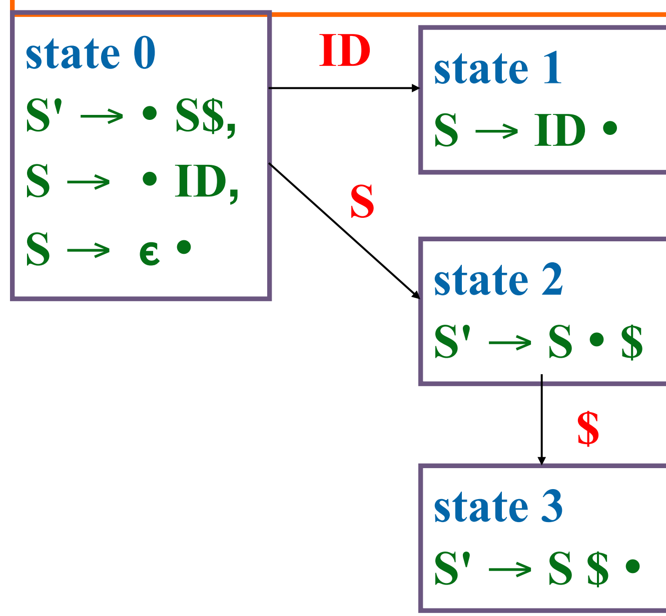
# Construction de la Table LR(0)(5)

- On obtient un machine à caractère d’état fini  (Characteristic finite state machine (CFSM))
  - il s’agit d’un automate fini
  - les ensemble de configuration et operation successeur sont déterminés par les état d'un CFSM et leurs transitions

```
void_build_CFSM(void)
{
  S = SET_OF(S0);
  while (S is nonempty) {
    Remove a configuration set s from S;
   /* Consider both terminals and non-terminals */
   for ( X in Symbols) {
     if(go_to0(s,X) does not label a CFSM state) {
       Create a new CFSM state and label it
         with go_to0(s , X) into S;
     }
     Create a transition under X from the state s
        labels to the state go_to0(s , X)
   }
  }
}
```

EX: for grammar $G_1$ :

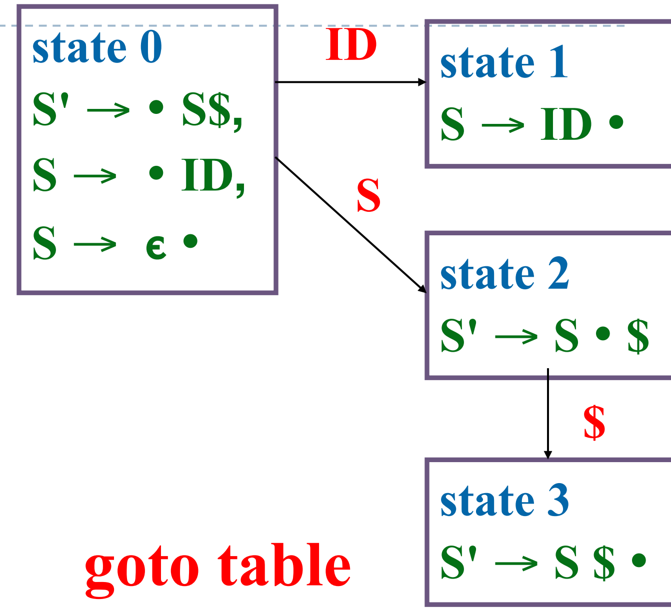
1. $S' \rightarrow S\$$
2. $S \rightarrow ID | \epsilon$



# Construction de la Table LR(0)(6)

- **CFSM is the goto table of LR(0) parsers.**

```
Int ** build_go_to_table(finite_automation CFSM)
{
  const int N = num_states (CFSM);
  int **tab;
  Dynamically allocate a table of dimension
    N × num_symbols (CFSM) to represent
    the go_to table and assign it to tab;
  Number the states of CFSM from 0 to N-1,
    with the Start State labeled 0;
  for( S = 0 ; S<=N-1 ; S++) {
  /* Consider both terminals and non-terminals. */
    for ( X in Symbols) {
      if ( State S has a transition under X to some state T)
        tab [S][X] = T ;
      else
        tab [S][X] = EMPTY;
    }
  }
  return tab;
}
```




**goto table**

| State | Symbol | | |
|---|---|---|---|
| | ID | $ | S |
| 0 | 1 | 4 | 2 |
| 1 | 4 | 4 | 4 |
| 2 | 4 | 3 | 4 |
| 3 | 4 | 4 | 4 |

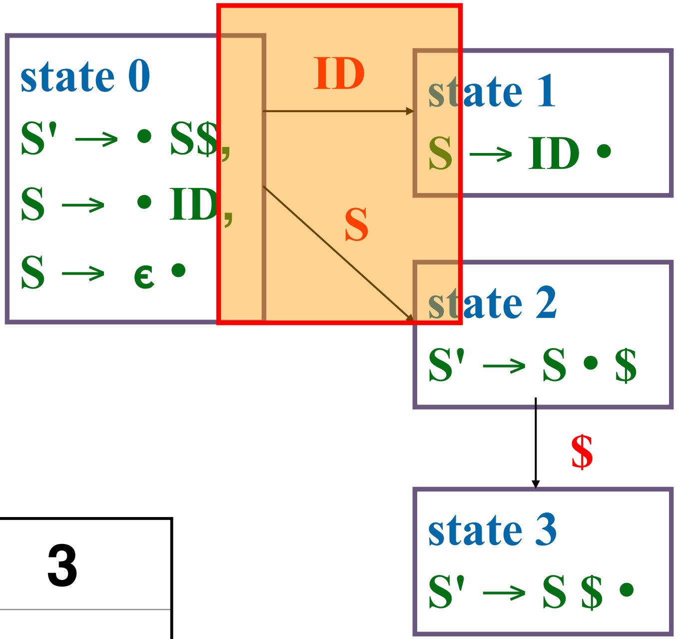
# Construction de la Table LR(0)(7)

- **Puisque LR(0) n'utilise aucun symbole d'anticipation, nous devons extraire la table action directement de l'ensemble de configuration du machine à caractère d'état fini CFSM**

# Construction de la Table LR(0)(8)

EX: for grammar $G_1$ :

1. $S' \rightarrow S\$$
2. $S \rightarrow ID | \epsilon$

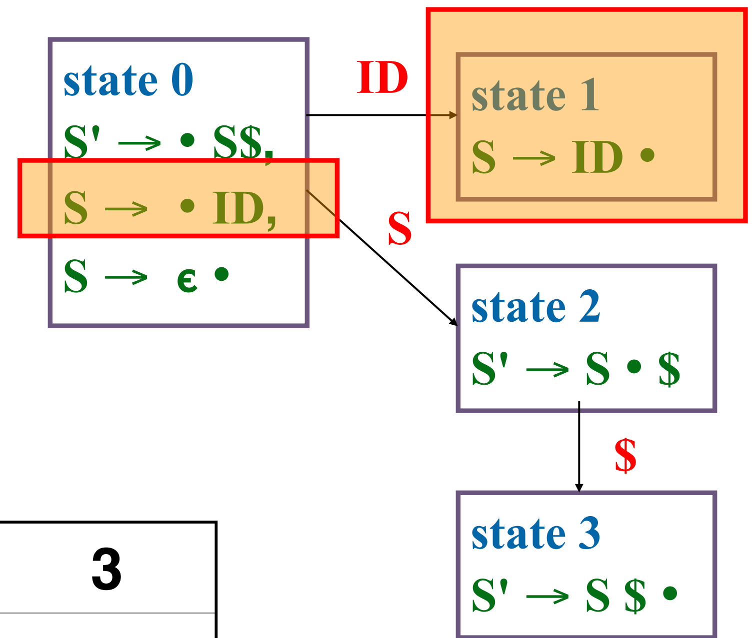



| state | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| action | S | R2 | S | Accept |

# Construction de la Table LR(0)(9)

EX: for grammar $G_1$ :

1. $S' \rightarrow S\$$
2. $S \rightarrow ID | \epsilon$

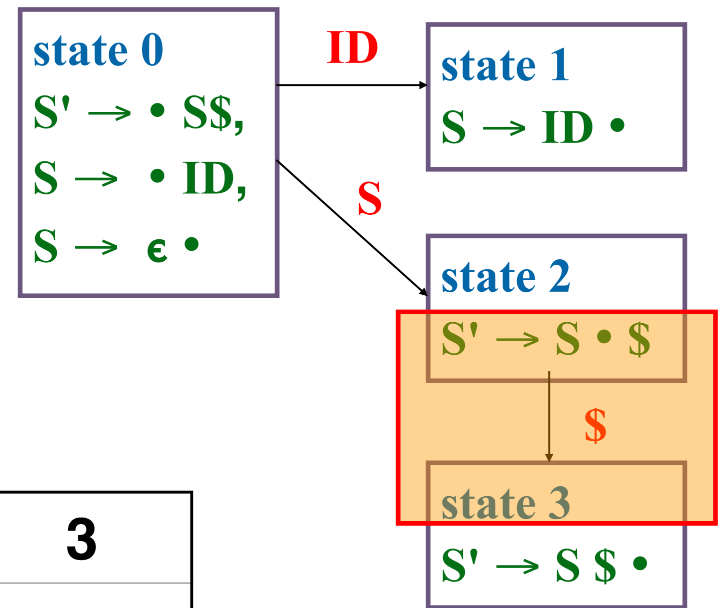



| state | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| action | S | R2 | S | Accept |

# Construction de la Table LR(0)(10)

EX: for grammar $G_1$ :

1. $S' \rightarrow S\$$
2. $S \rightarrow ID | \epsilon$

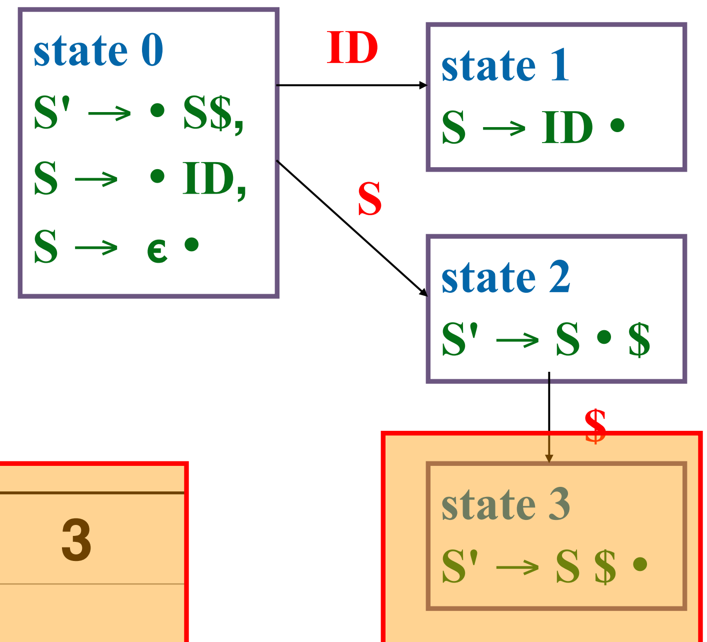



| state | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| action | S | R2 | S | Accept |

# Construction de la Table LR(0)(11)

**EX: for grammar $G_1$ :**

1. $S' \rightarrow S\$$
2. $S \rightarrow ID | \epsilon$




| state | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| action | S | R2 | S | Accept |

# Exemple de Trace LR(0) « (id)$ » (0)

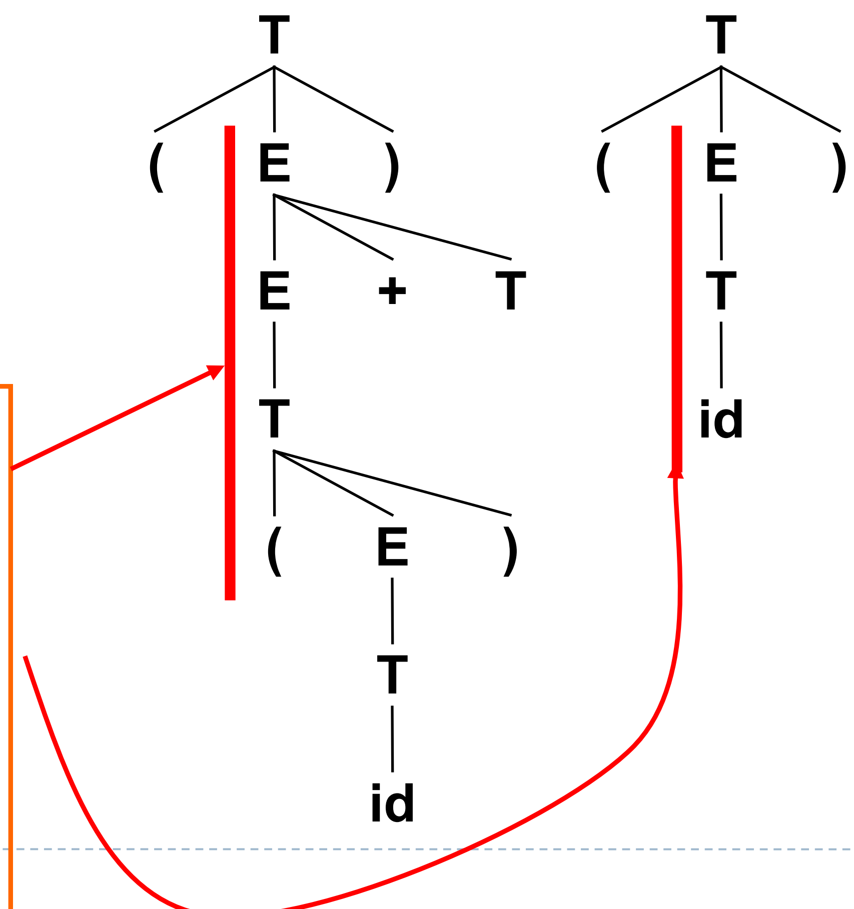
- **Avant de tracer nous devons connaitre l'intuition du CFSM**

**for grammar $G_2$ :**

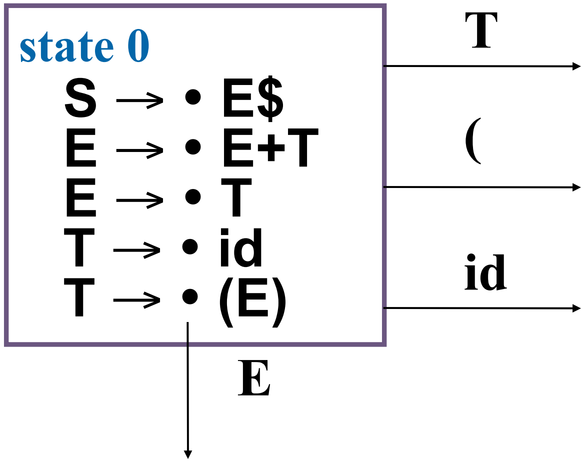
1. $S \rightarrow E\$$
2. $E \rightarrow E+T$
3. $E \rightarrow T$
4. $T \rightarrow id$
5. $T \rightarrow (E)$

- closure0( { $T \rightarrow (\bullet\ E\ )$ } 
- = { $T \rightarrow (\bullet\ E\ )$ ,
  $E \rightarrow \bullet\ E + T$ ,
  $E \rightarrow \bullet\ T$ ,
  $T \rightarrow \bullet\ id$ ,
  $T \rightarrow \bullet\ (\ E\ )$ }

Quand on fait **shift** '(' , qqn arbres possibles :



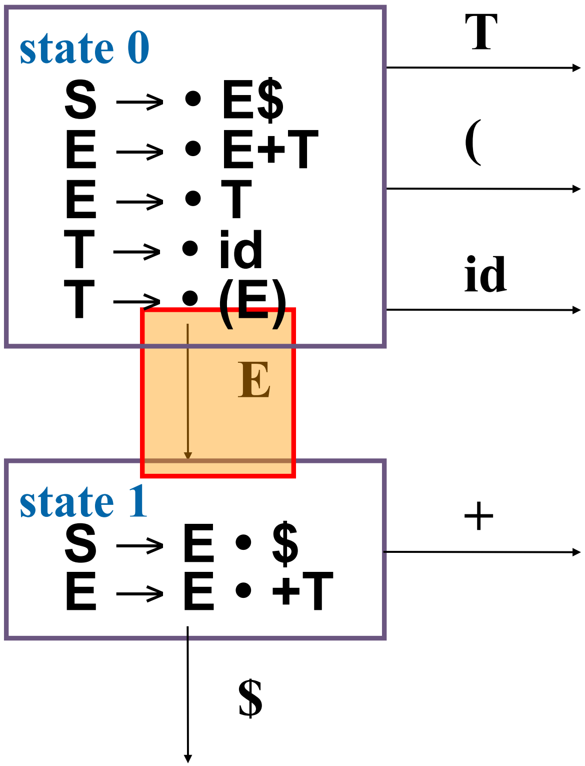
# Exemple de Trace LR(0) (1)




$$closure0(\{\, S \rightarrow \bullet\, E\$ \,\}) =$$

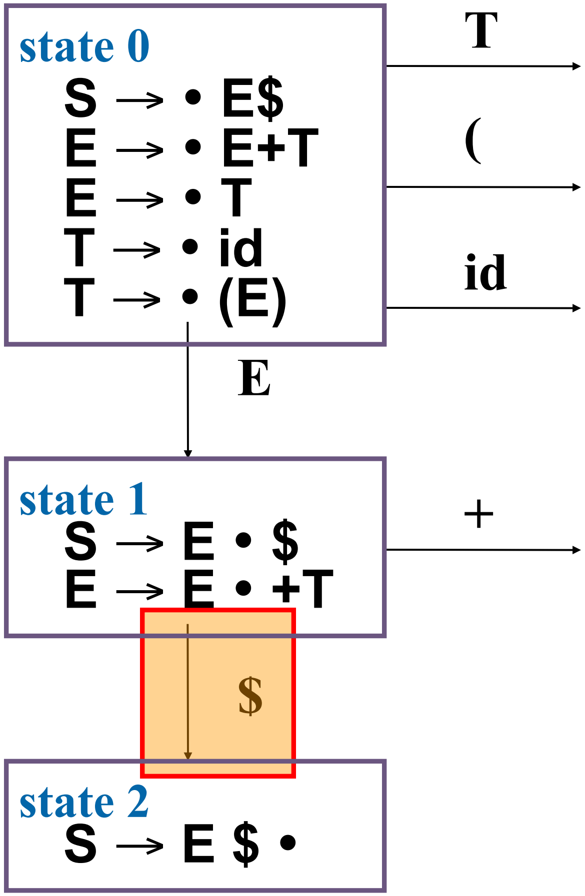
$$\begin{aligned}\{\ & S \rightarrow \bullet\, E\$, \\ & E \rightarrow \bullet\, E{+}T, \\ & E \rightarrow \bullet\, T, \\ & T \rightarrow \bullet\, id, \\ & T \rightarrow \bullet\, (E) \quad \}\end{aligned}$$

# Exemple de Trace LR(0) (2)



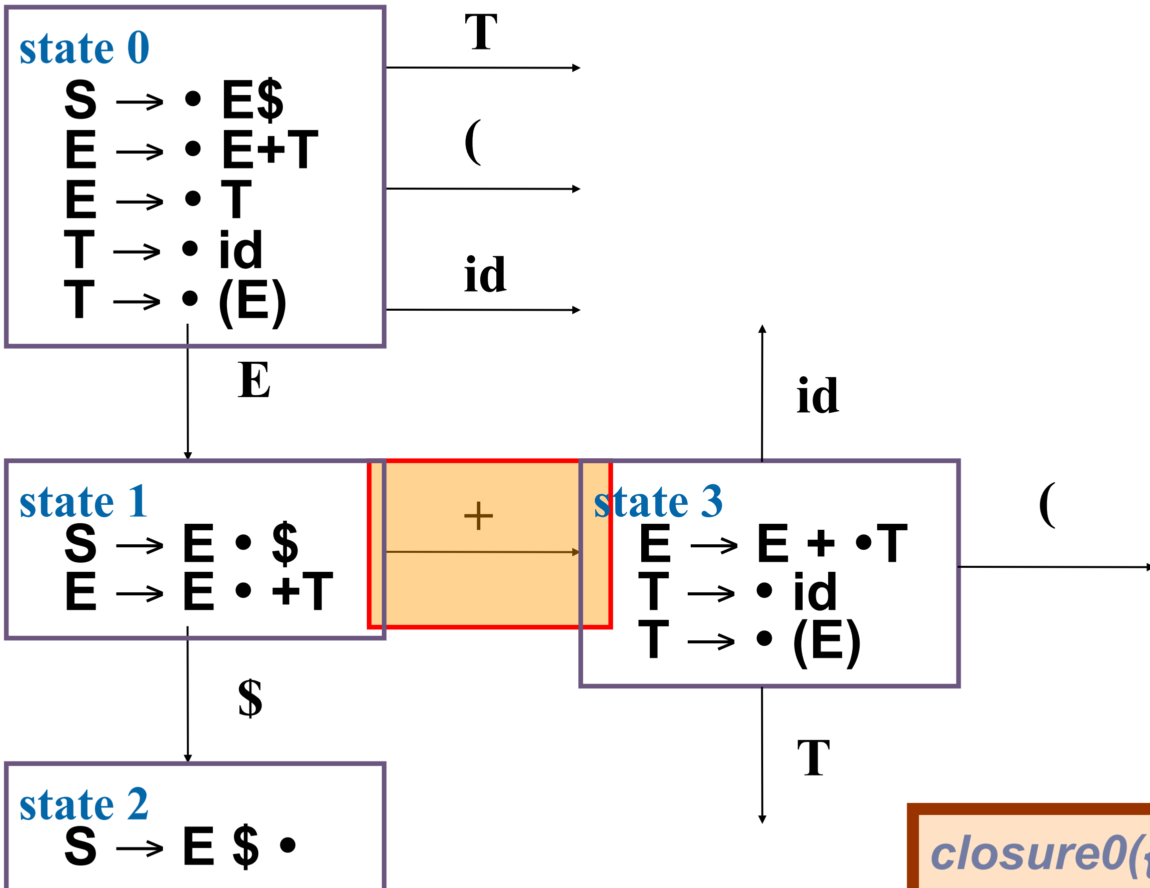

*closure0({ S → E • $, E → E • +T } )*
*=itself*

# Exemple de Trace LR(0) (3)



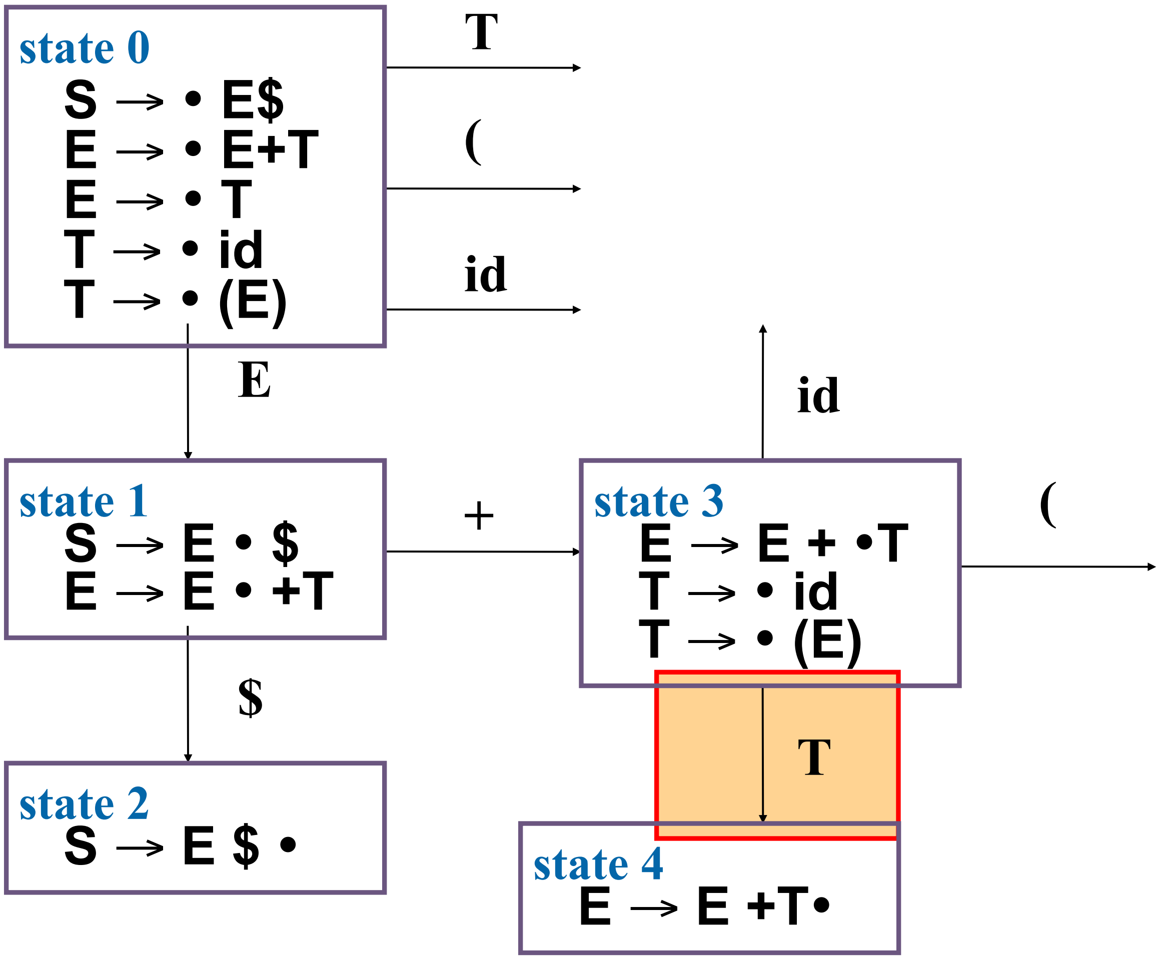

*closure0({ S → E $ •} ) =itself*

# Exemple de Trace LR(0) (4)



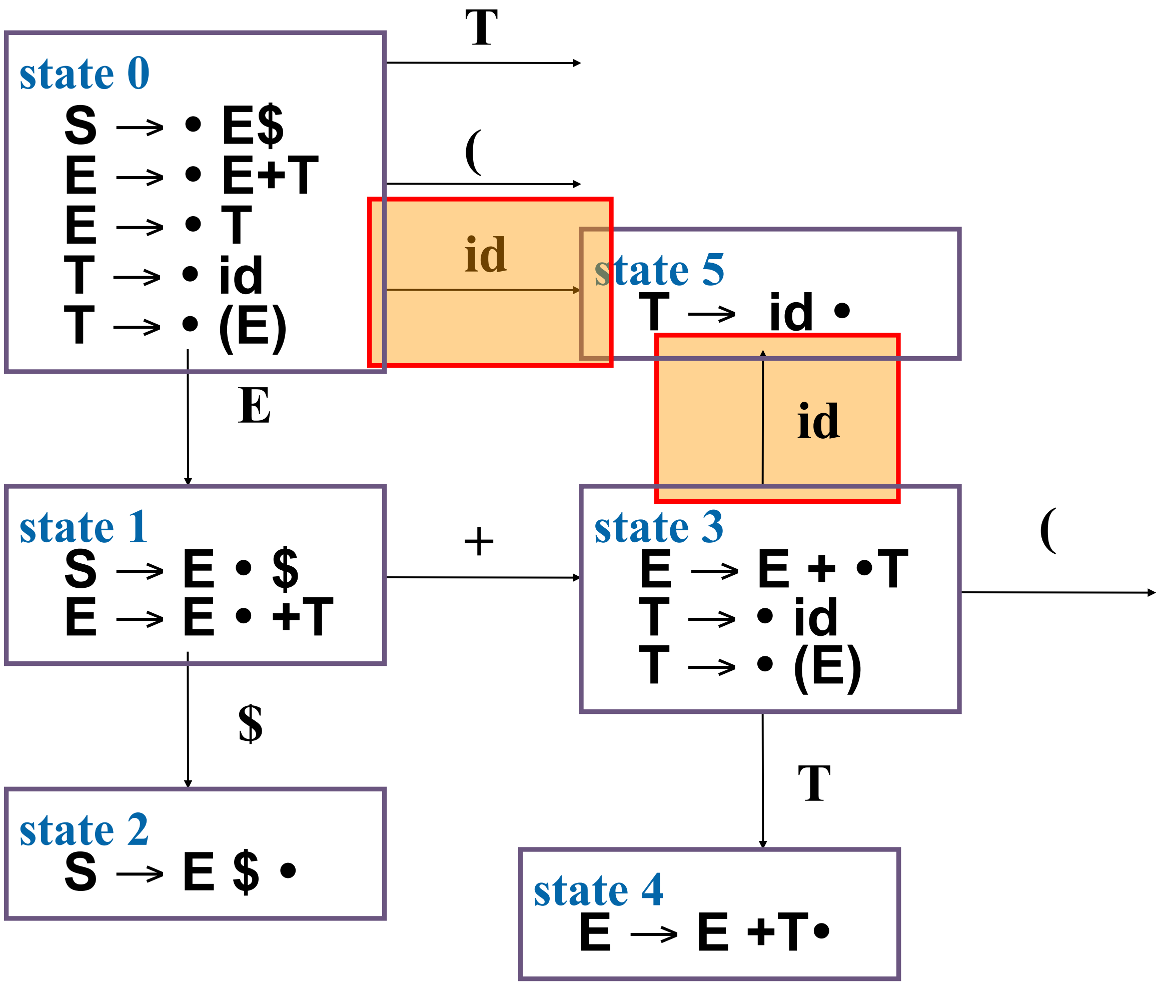

*closure0({E → E+ • T}) =*
*{E → E+ • T,*
*T → • id,*
*T → • (E) }*

# Exemple de Trace LR(0) (5)



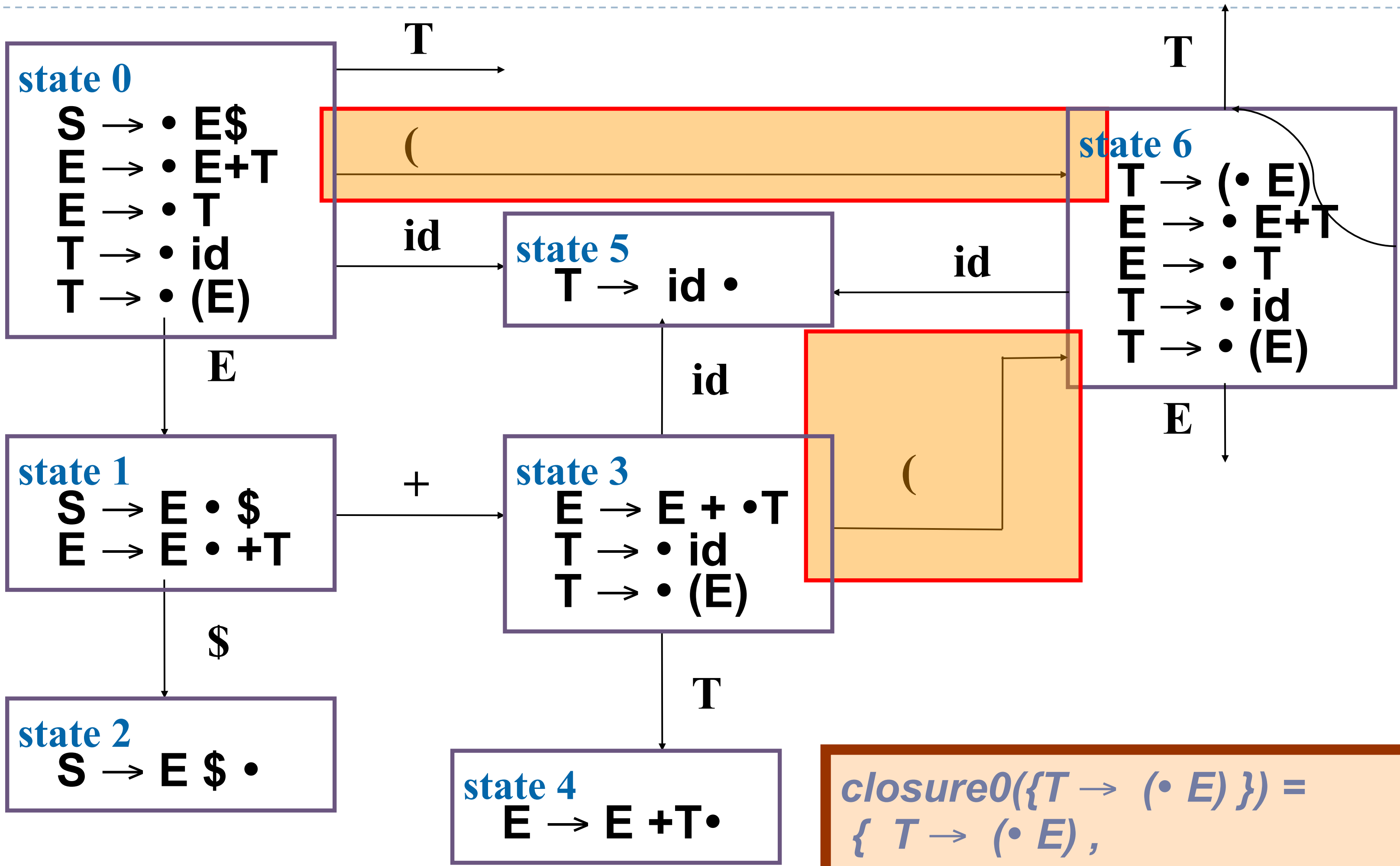

*closure0({E → E+ T • }) =itself*

# Exemple de Trace LR(0) (6)



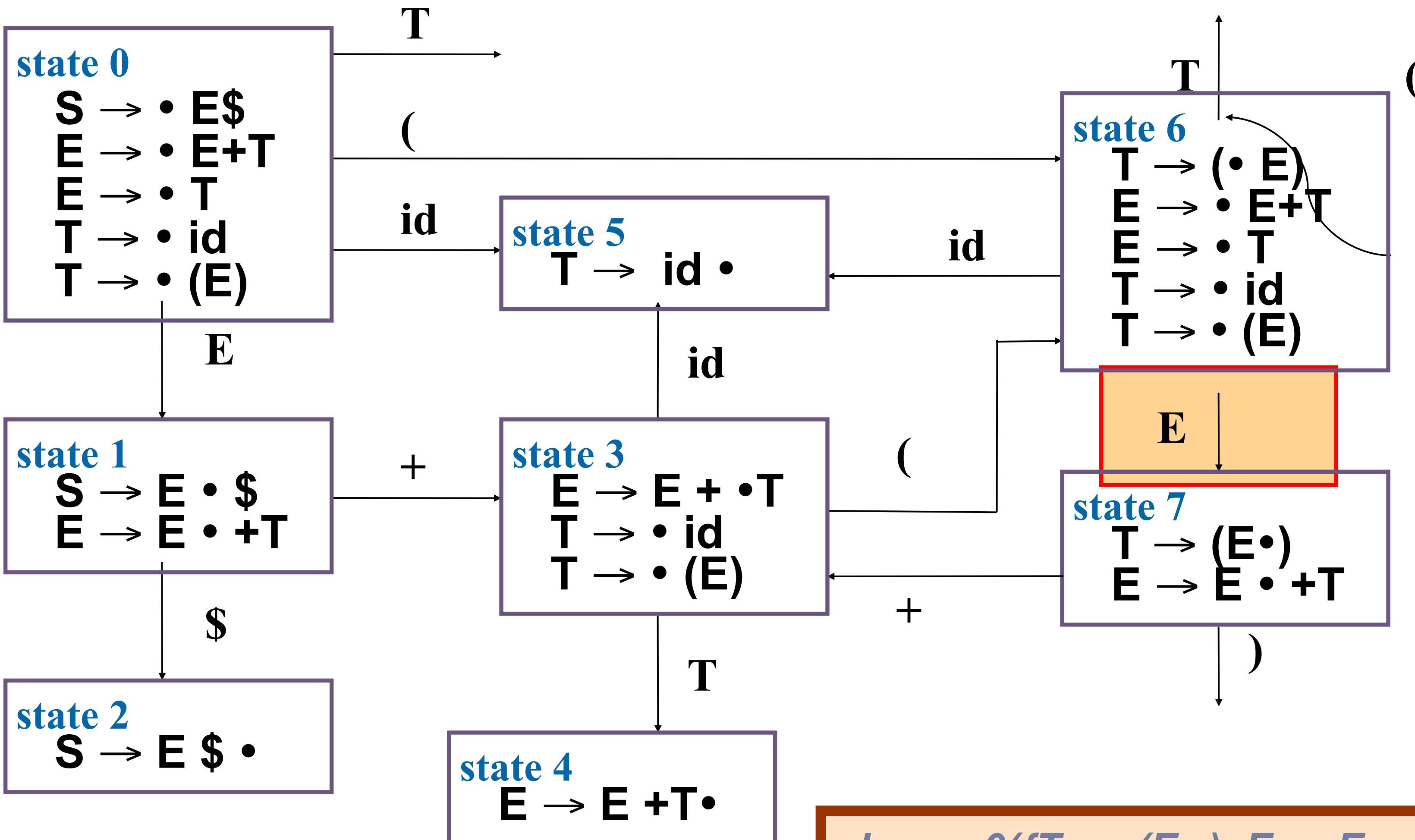

*closure0({T → id • }) =itself*

# Exemple de Trace LR(0) (7)



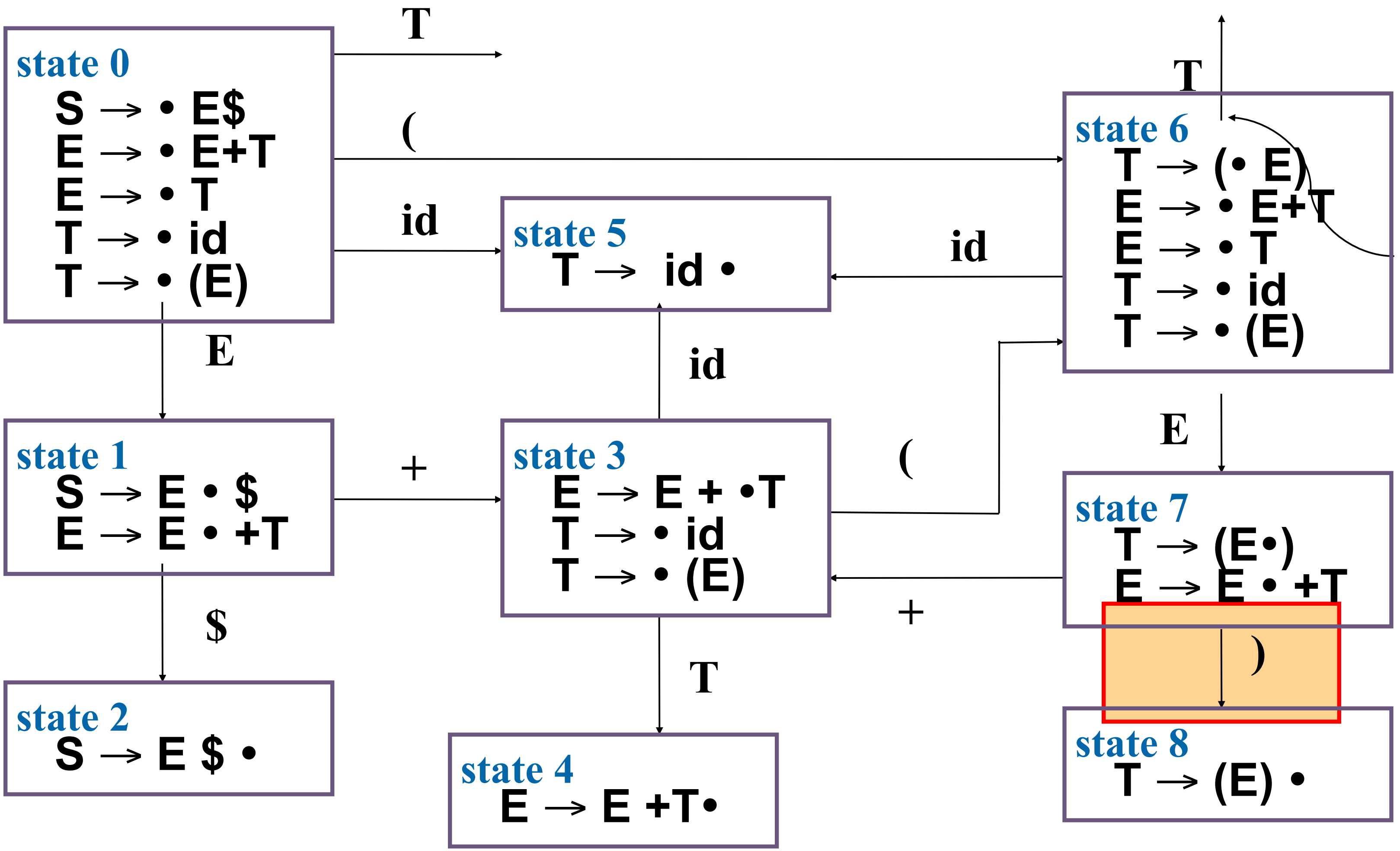

*closure0({T → (• E) }) =*
*{ T → (• E) ,*
*E → • E+T,*
*E → • T,*
*T → • id,*
*T → • (E) }*

# Exemple de Trace LR(0) (8)



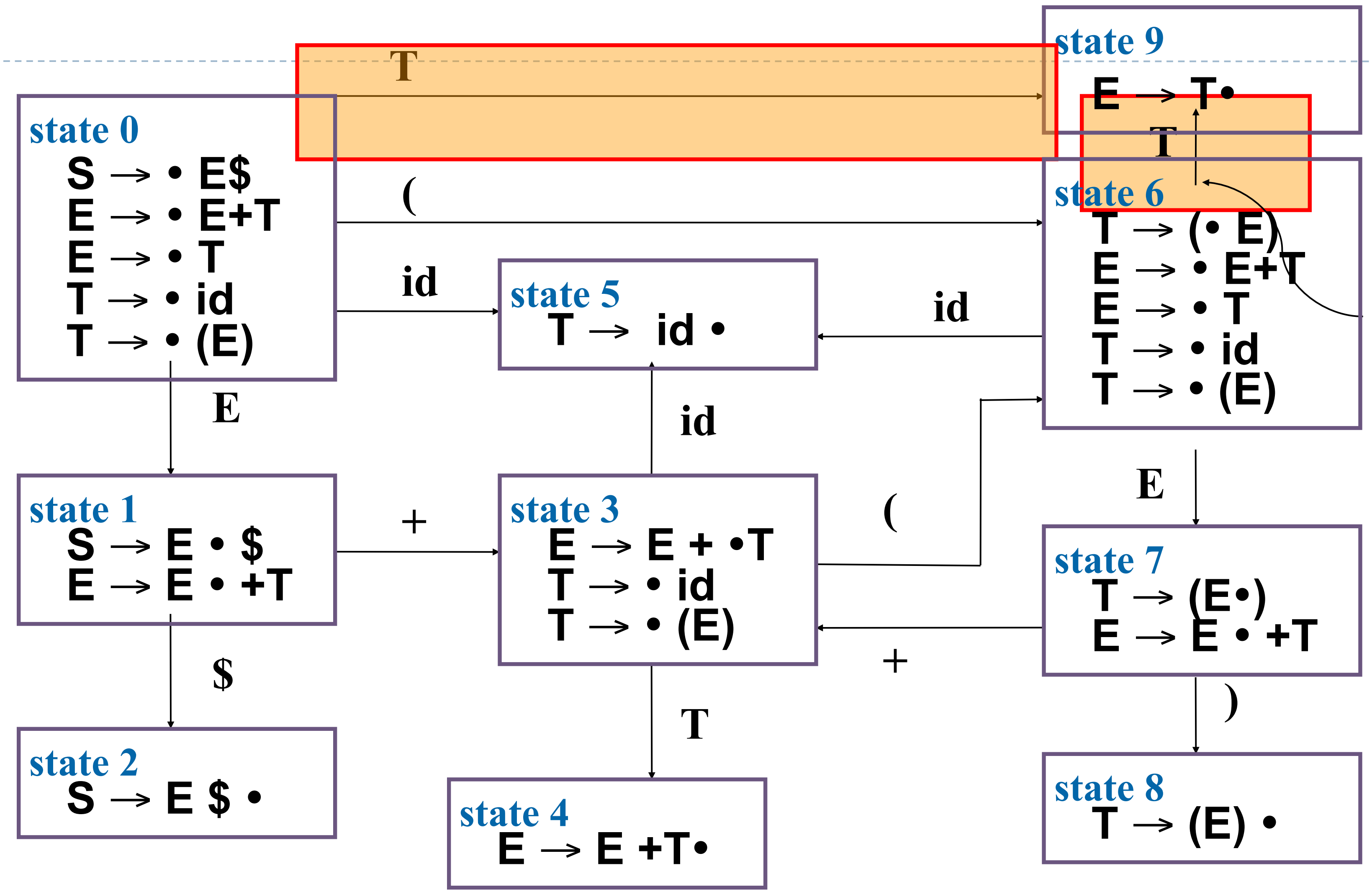

*closure0({T → (E• ) ,E → E • +T } )*
*=itself*

# Exemple de Trace LR(0) (9)



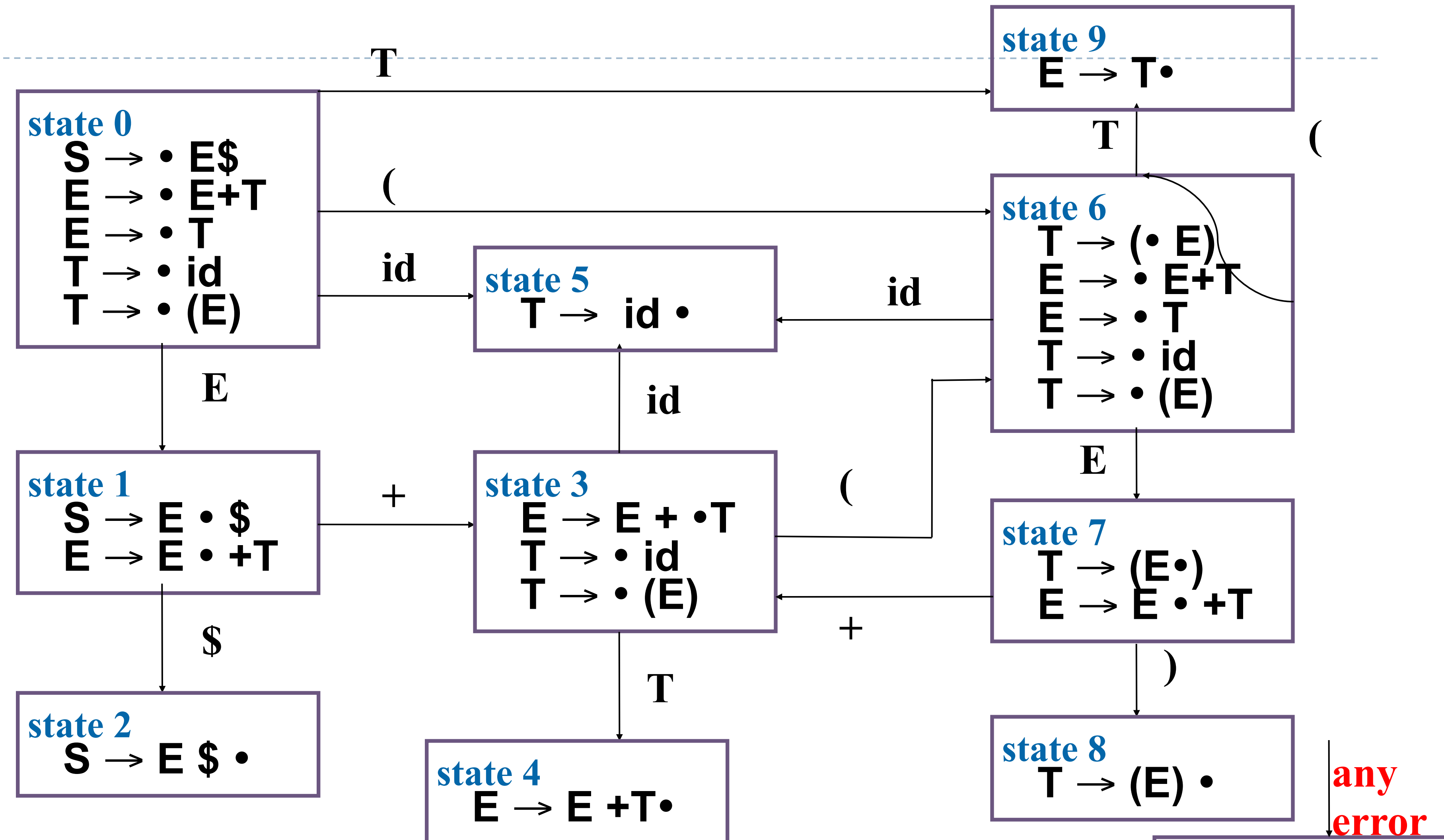

closure0({T → (E ) •} ) =itself

# Exemple de Trace LR(0)(10)




*closure0({E → T •} ) =itself*

# Exemple de Trace LR(0)(11)

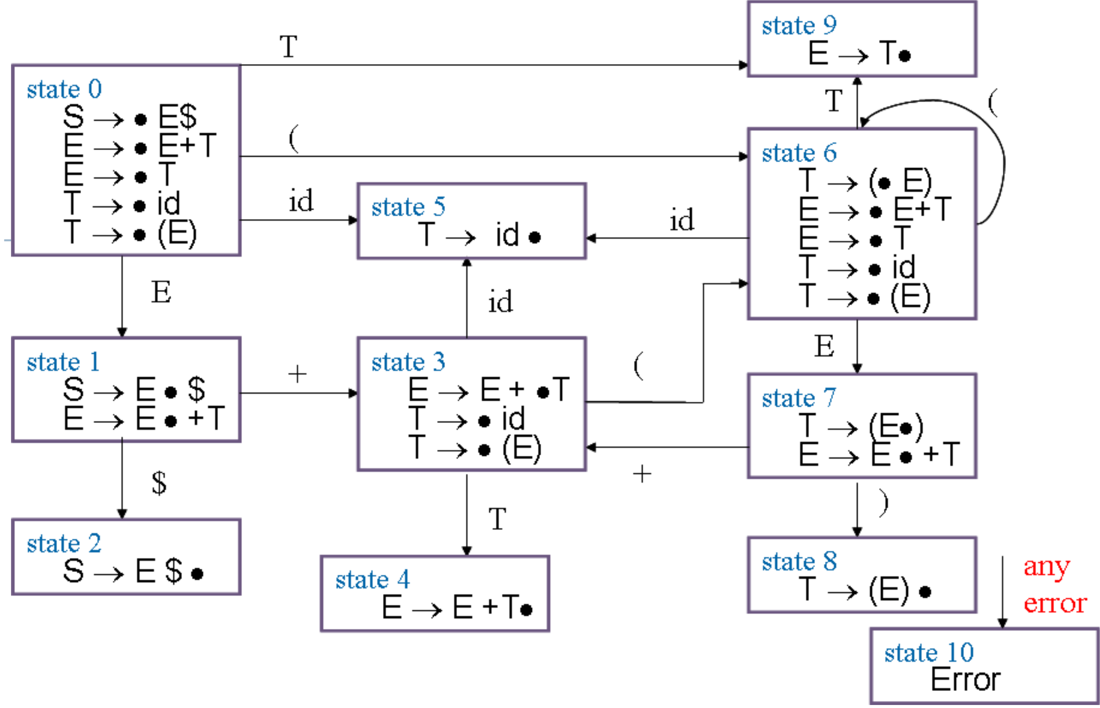



action table

| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

# Exemple de Trace LR(0)(12)

| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |




goto table

| State | S | E | T | + | id | ( | ) | $ |
|---|---|---|---|---|---|---|---|---|
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

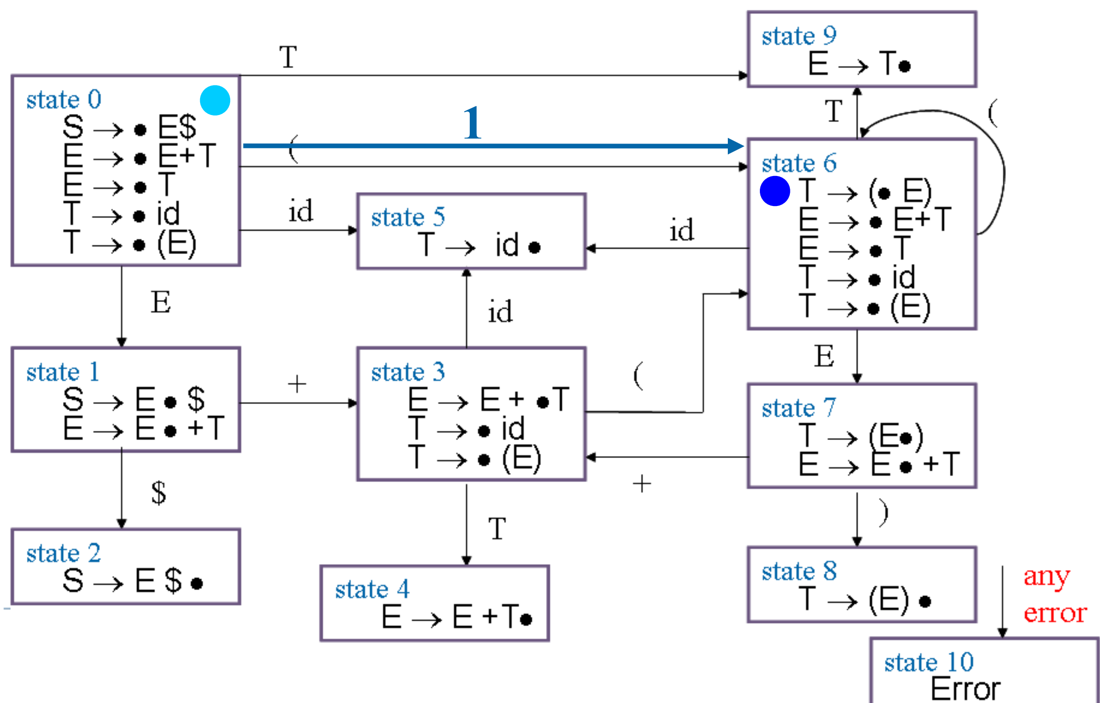
# Program Example (1)

## Initial :(id)$

**step1:0 (id)$ shift (**

**Tree:**

(




| Symbol | State 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

# Program Example (2)

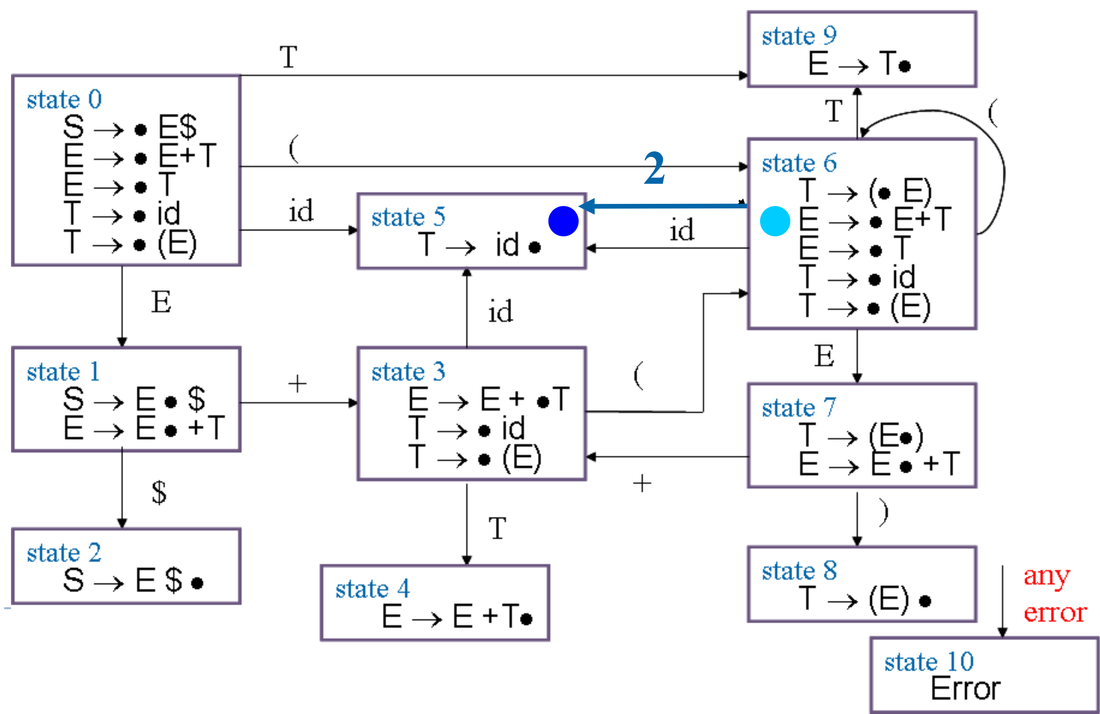
## Initial :(id)$

| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

**step2:06** id)$ **shift id**

**Tree:**

( id




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

# Program Example (3)

## Initial :(id)$

| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

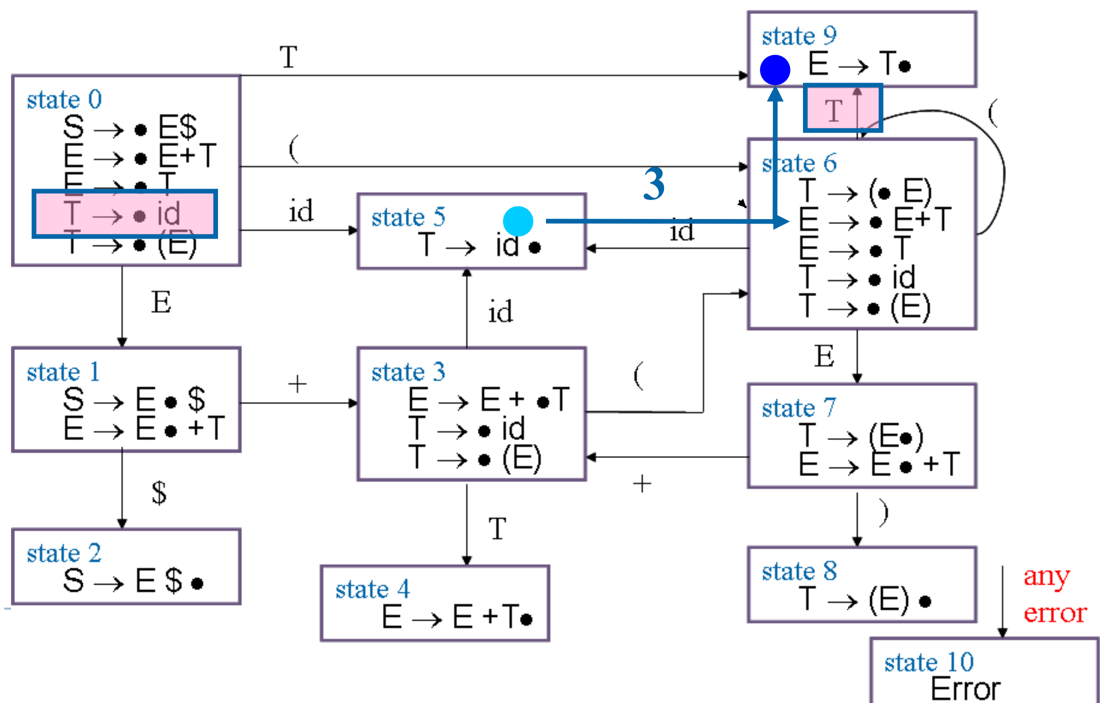
**step3:065** )$ **reduce 4**

**Tree:**

( T

|

id




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

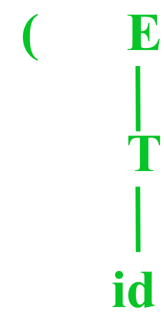
# Program Example (4)

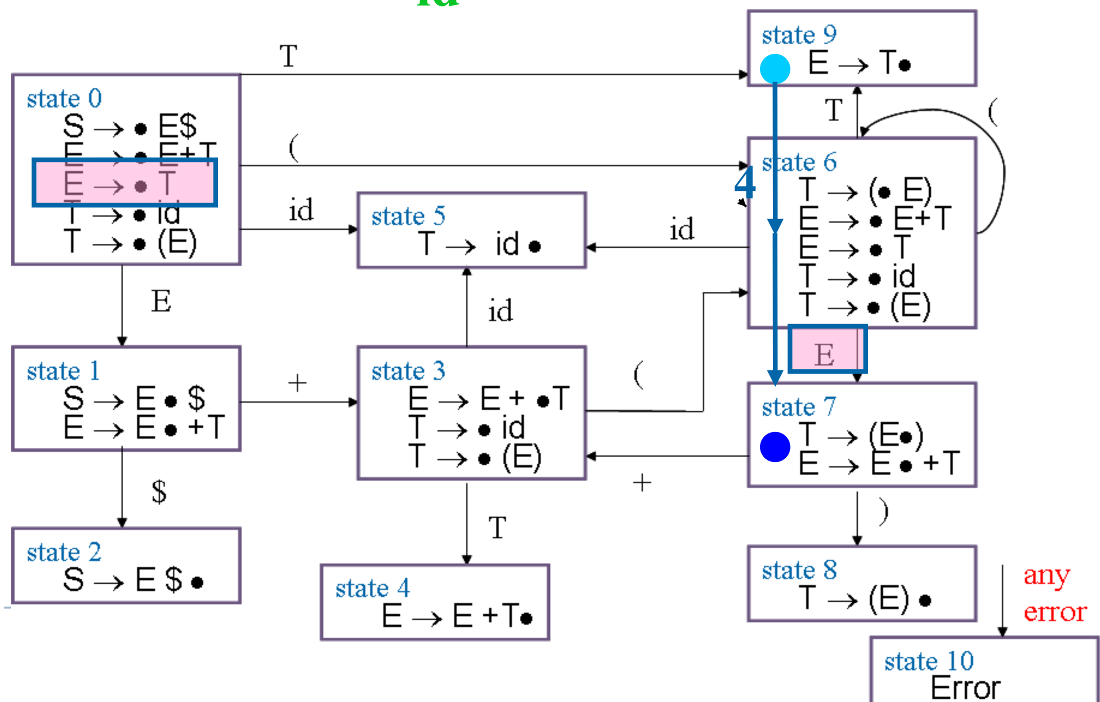
## Initial :(id)$

| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

step4:069 )$ reduce 3

Tree:




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

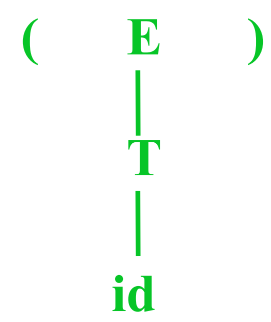
# Program Example (5)

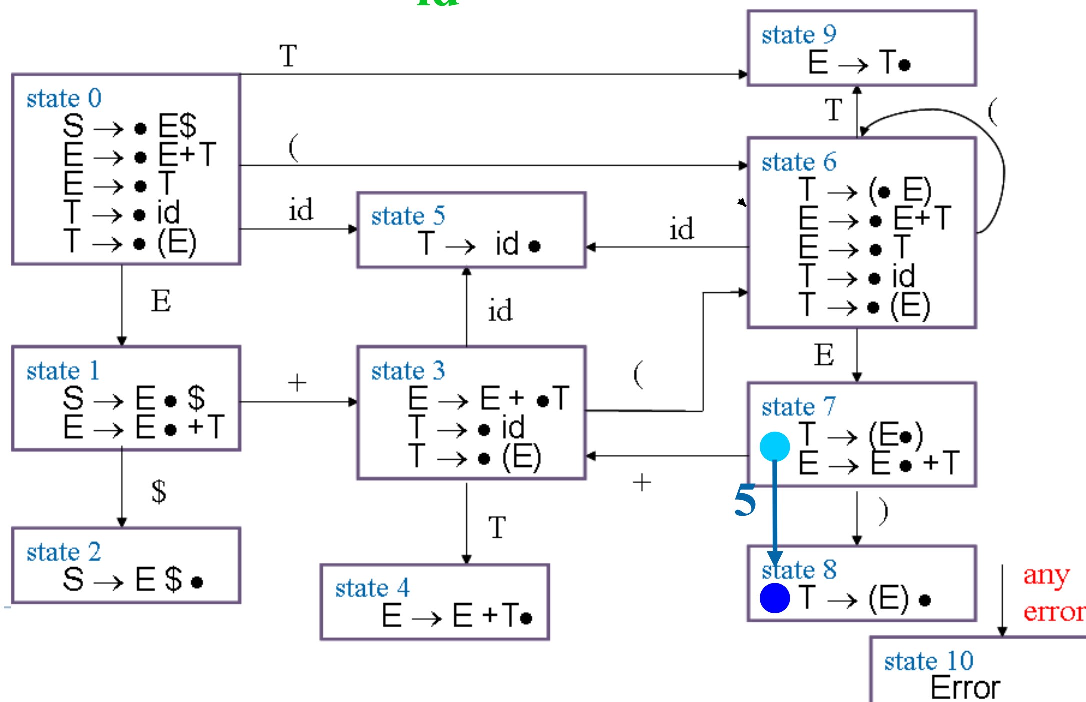
## Initial :(id)$

| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

**step5:067 )$ shift )**

**Tree:**




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

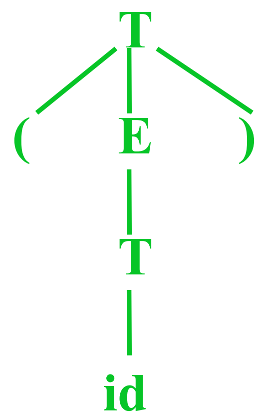
| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

# Program Example (6)

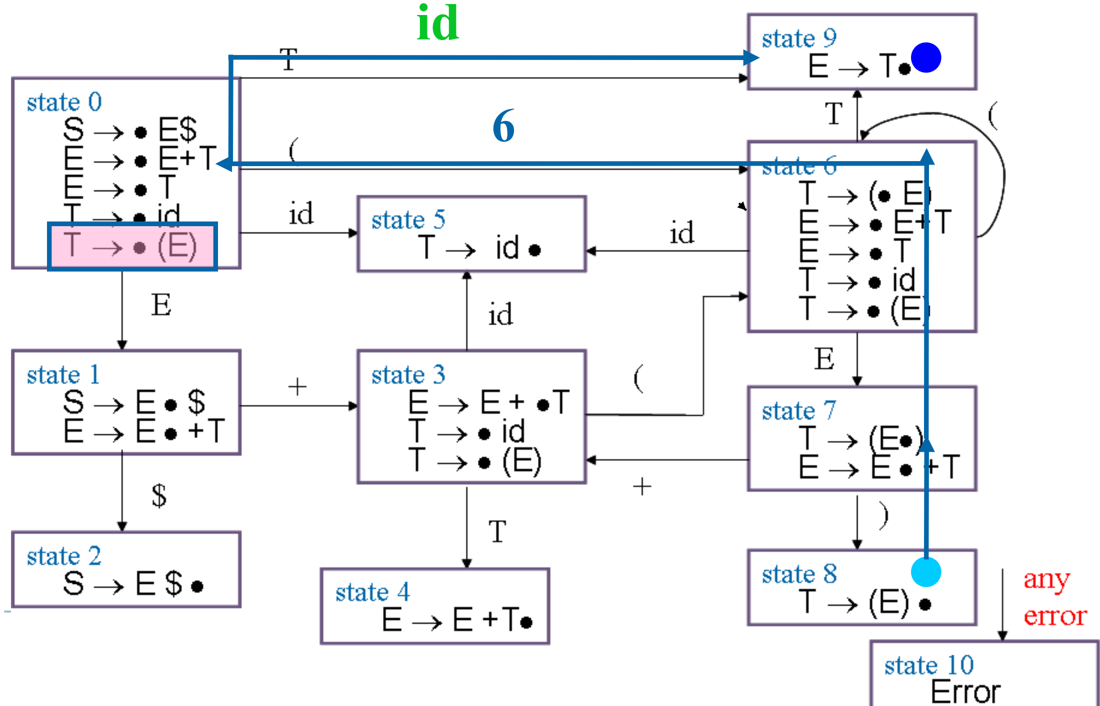
## Initial :(id)$

**step6:0678 $ reduce 5**

**Tree:**




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |



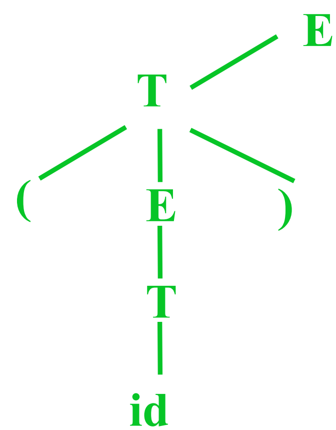
# Program Example (7)

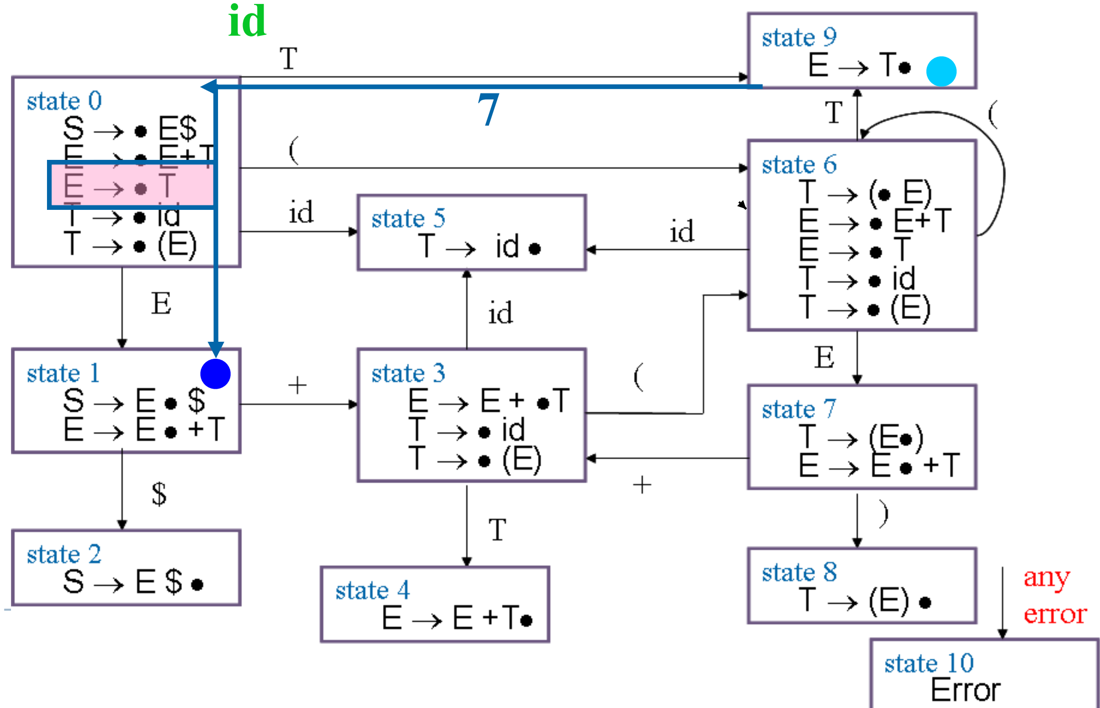
## Initial :(id)$

| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

step7:09 $ reduce 3

Tree:




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

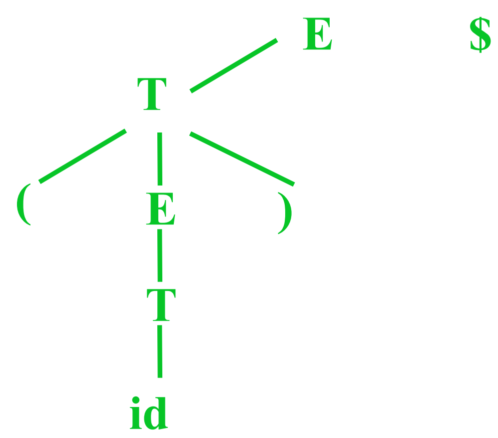
# Program Example (8)

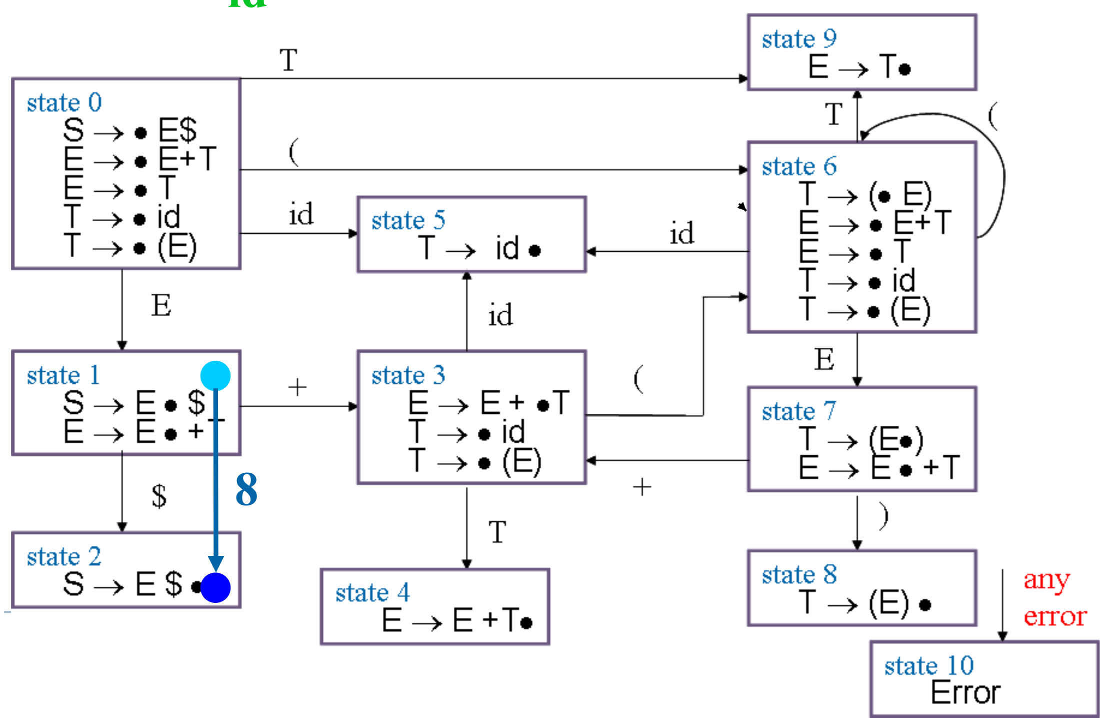
## Initial :(id)$

| State | S | E | T | + | id | ( | ) | $ |
|---|---|---|---|---|---|---|---|---|
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

**step8:01** $ shift $

**Tree:**




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |

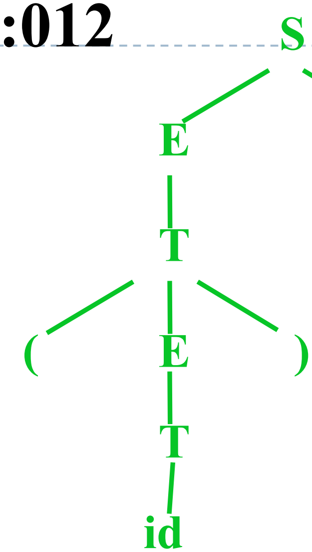
# Program Example (9)

## Initial :(id)$

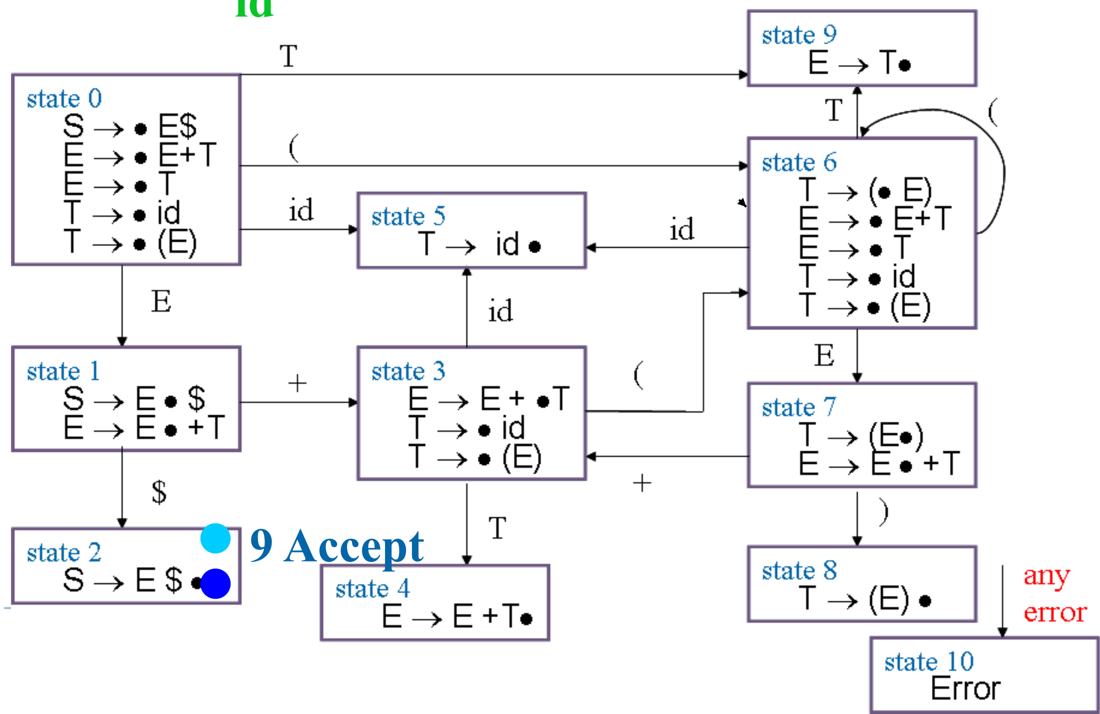
**step9:012** **Accept**

| State | Symbol | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S | E | T | + | id | ( | ) | $ |
| 0 | | 1 | 9 | | 5 | 6 | | |
| 1 | | | | 3 | | | | 2 |
| 2 | | | | | | | | |
| 3 | | | 4 | | 5 | 6 | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | 7 | 9 | | 5 | 6 | | |
| 7 | | | | 3 | | | 8 | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |

Tree:




| Symbol | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| anything | S | S | A | S | R2 | R4 | S | S | R5 | R3 | |



# Exercice

for grammar $G_3$ :

1. $S \rightarrow E \ \$$
2. $E \rightarrow E + T$
3. $E \rightarrow T$
4. $T \rightarrow T * P$
5. $T \rightarrow P$
6. $P \rightarrow id$
7. $P \rightarrow ( E )$

# Question 2

**Quel est le problème avec LR(0) sur $G_3$ ?**

# Plan

- Introduction
- Analyseur à décalage-reduction (Shift-Reduce Parsers)
- LR Parsers
- LR(1) Parsing
- SLR(1)Parsing
- LALR(1