Parsing Ascendente (bottom-up)

- Constrói a árvore de parsing das folhas para a raiz.
- É mais complexo de que o parsing descendente (top-down), e quase ninguém o implementa manualmente.

  → usam um gerador que constrói o parser automáticamente a partir da gramática (ex: yacc/bison)

- Mais genérico que um parser LL(1) porque não necessita de ter a gramática sem recursividade à esquerda e factorizada à esquerda

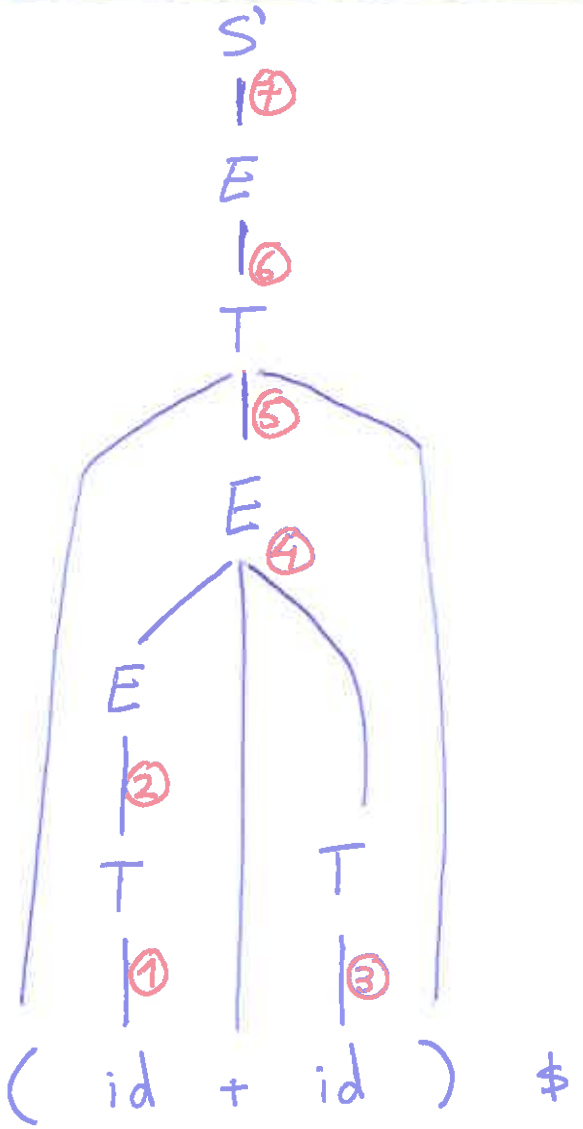
Os exemplos e notas que se seguem estão baseados nas notas de Maggie Johnson e Julie Zelenski (ver link a partir da página web da disciplina) e também no livro do dragão.

Exemplo

$S' \rightarrow E$
$E \rightarrow T$
$E \rightarrow E + T$
$T \rightarrow id$
$T \rightarrow (E)$

$ significa o fim do input

Input: (id + id) $



Derivação mais à direita por ordem inversa.

1. $T \Rightarrow id$
2. $E \Rightarrow T$
3. $T \Rightarrow id$
4. $E \Rightarrow E + T$
5. $T \Rightarrow E$
6. $E \Rightarrow T$
7. $S' \Rightarrow E$

- A partir da gramática é construído uma tabela de parsing. Na realidade são duas tabelas chamadas Action e Goto.

- Veremos como se constroem estas tabelas mais adiante.

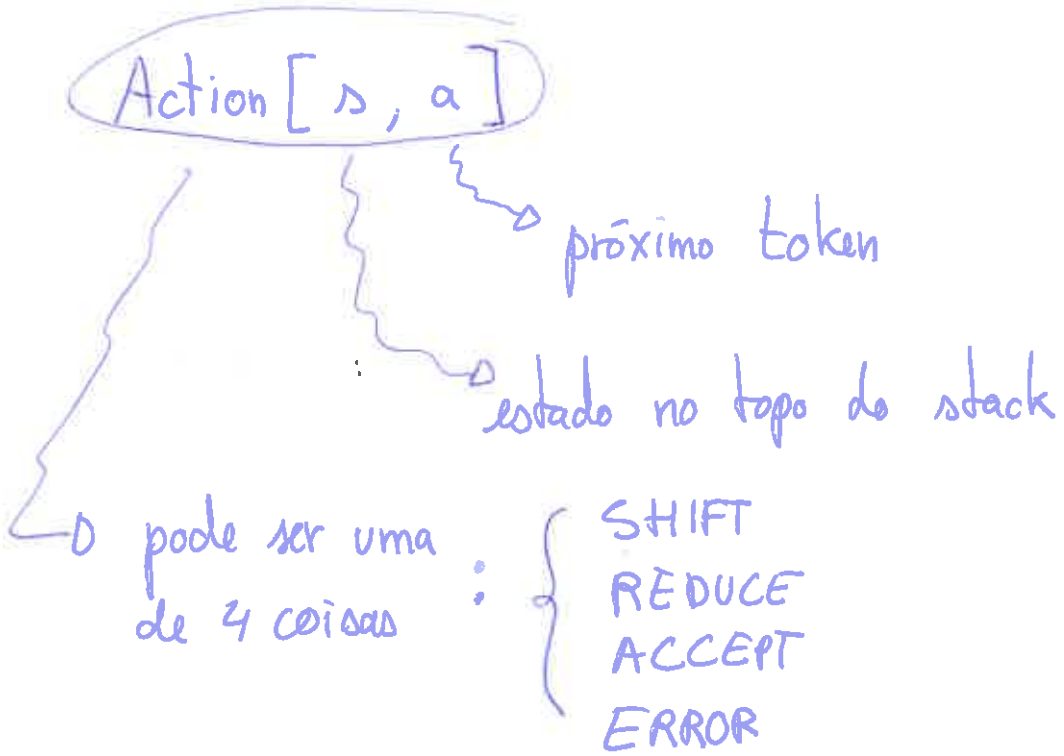
- O parsing é feito lendo o input da esquerda para a direita, e fazendo uma série de shifts (push num stack) e reduções (pops no stack).

- Uma redução consiste em reconher os símbolos que estão no lado direito de uma regra da gramática e substituí-los pelo símbolo que está no lado esquerdo da regra.

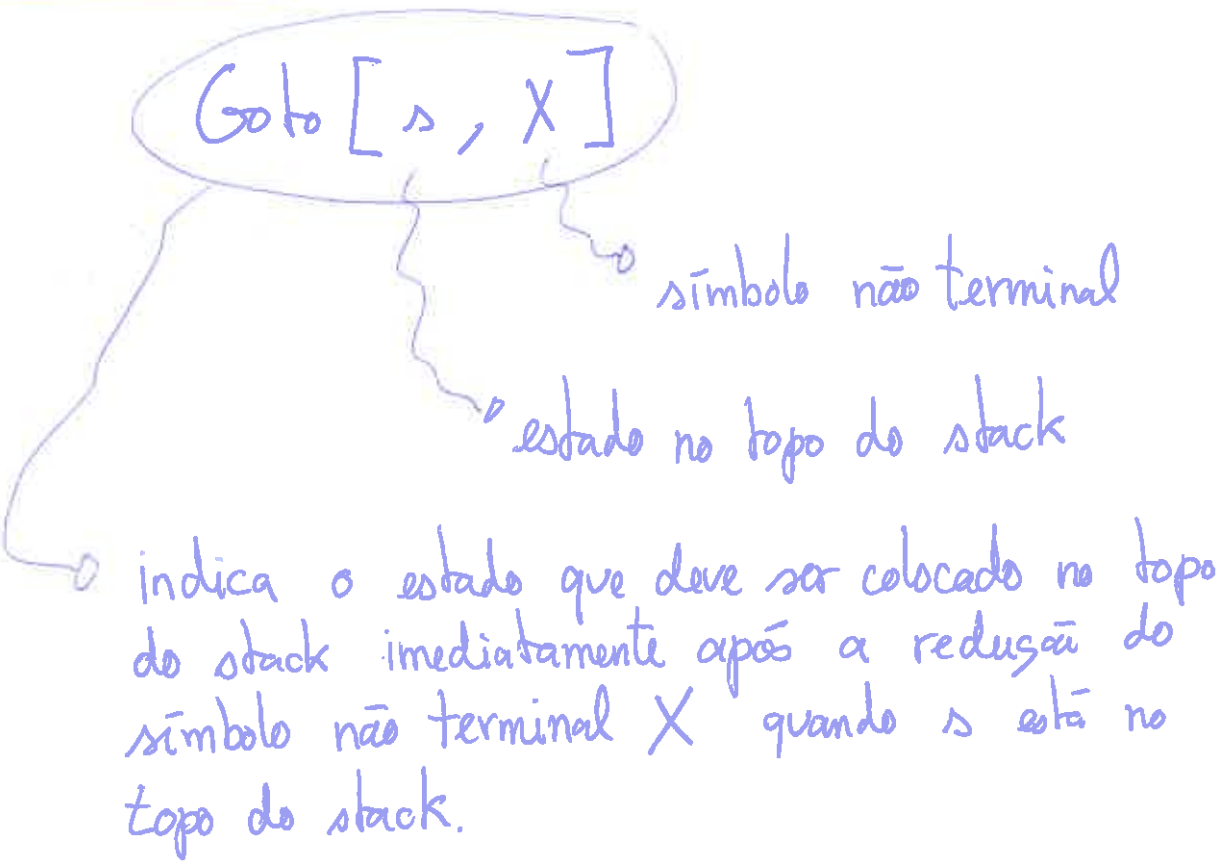
- O stack contém estados. Cada estado representa uma possível situação em que o parser pode estar.

- O parser tem 4 acções possíveis:
  shift, reduce, error, accept.

Action Table



Goto Table



Método

- Começar com o estado inicial s_0 no stack.
- Se o próximo token for a e o estado no topo do stack for s_t, então:
  - Se Action$[s_t, a]$ for shift n, fazemos push do estado n e avançamos o input para o próximo token.
  - Se Action$[s_t, a]$ for reduce $Y \to X_1 \dots X_k$ fazemos pop de k estados e vemos qual o estado que fica no topo do stack. Designemos esse estado por s_u.
    Goto$[s_u, Y]$ indica qual o próximo estado que deve ser colocado no topo do stack. Isto é, fazemos push(Goto$[s_u, Y]$)
  - Se Action$[s_t, a]$ for accept, o parsing é aceite.
  - Se Action$[s_t, a]$ for error, temos um erro de sintaxe e o parsing não é aceite.

Exemplo

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

shift estado 4

reduce usando a regra 2)

número do estado para ser empilhado

Estado no topo do Stack	Action					Goto	
	id	+	(	)	$	E	T
0	$\Delta 4$		$\Delta 3$			1	2
1		$\Delta 5$			Accept		
2	r2	r2	r2	r2	r2		
3	$\Delta 4$		$\Delta 3$			6	2
4	r4	r4	r4	r4	r4		
5	$\Delta 4$		$\Delta 3$				8
6		$\Delta 5$		$\Delta 7$			
7	r3	r3	r3	r3	r3		
8	r1	r1	r1	r1	r1		

~~> entradas vazias correspondem a Erro

Trace do parsing de (id + id) usando as tabelas Action e Goto.

Stack	Símbolos	Input	Acção do Parser
S_0		(id + id) $	shift S_3
$S_0 S_3$	(	id + id) $	shift S_4
$S_0 S_3 S_4$	(id	+ id) $	reduce 4: T → id Goto S_2
$S_0 S_3 S_2$	(T	+ id) $	reduce 2: E → T Goto S_6
$S_0 S_3 S_6$	(E	+ id) $	shift S_5
$S_0 S_3 S_6 S_5$	(E +	id) $	shift S_4
$S_0 S_3 S_6 S_5 S_4$	(E + id	) $	reduce 4: T → id Goto S_8
$S_0 S_3 S_6 S_5 S_8$	(E + T	) $	reduce 1: E → E+T Goto S_6
$S_0 S_3 S_6$	(E	) $	shift S_7
$S_0 S_3 S_6 S_7$	(E)	$	reduce 3: T → (E) Goto S_2
$S_0 S_2$	T	$	reduce 2: E → T Goto S_1
$S_0 S_1$	E	$	Accept

Como obter as tabelas Action e Goto ?

- Como é que o parser sabe qual a decisão que deve tomar a cada instante?
- O parser guarda estados no stack para saber em que ponto do parsing está.
- Os estados representam conjuntos de "items".
- Um item é uma regra da gramática com um <u>ponto</u> numa determinada posição.

Ex: $A \rightarrow X\,Y\,Z$

Esta regra pode dar origem a 4 items:

$$\begin{cases} A \rightarrow \cdot\, X\,Y\,Z \\ A \rightarrow X \cdot Y\,Z \\ A \rightarrow X\,Y \cdot Z \\ A \rightarrow X\,Y\,Z \cdot \end{cases}$$

A regra $A \rightarrow \varepsilon$ só tem 1 item

$A \rightarrow \cdot$

- Um item indica qual a parte da regra que já foi vista pelo parsing.

Ex: $A \rightarrow X . YZ$

indica que já vimos no input uma sequência de tokens que foi derivada de X, e estamos à espera de ver no input uma outra sequência de tokens que sejam derivados a partir de YZ

$A \rightarrow XYZ .$

Indica que vimos uma seq. de tokens derivados de XYZ, e que poderá ser possível fazer uma redução de XYZ para A

- Fecho (em inglês, closure) de um conjunto de items I é um conjunto de items construídos a partir de I através das seguintes 2 regras:

  ① Inicialmente, todo o item de I faz parte de closure(I).

  ② Se $A \to \alpha . B\beta$ está em closure(I) e $B \to \gamma$ for uma regra da gramática, então acrescenta o item $B \to .\gamma$ a closure(I) caso ainda não lá esteja.
  Aplicar esta regra até não ser possível acrescentar nada a closure(I).

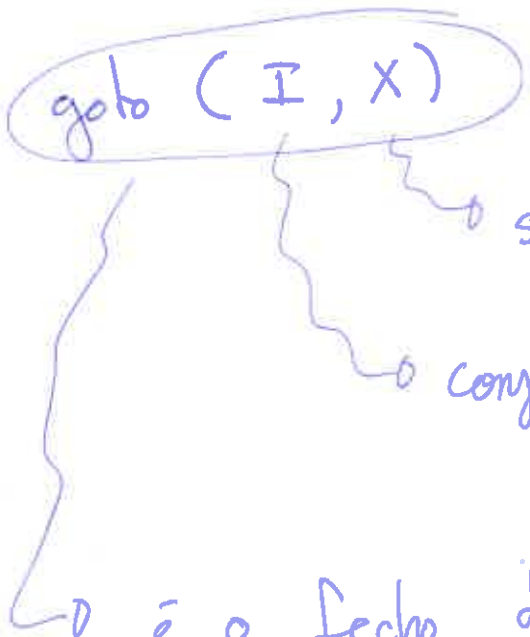
Exemplo

0) $S' \to E$
1) $E \to E+T$
2) $E \to T$
3) $T \to (E)$
4) $T \to id$

closure({ $S' \to .E$ }) =

$S' \to .E$ // regra ①
$E \to .E+T$ // regra ②
$E \to .T$ // regra ②
$T \to .(E)$ // regra ②
$T \to .id$ // regra ②

Operação goto



é o fecho do conjunto de todos os items $A \to \alpha X \cdot \beta$ tais que $A \to \alpha \cdot X\beta$ esteja em I.

Exemplo: mesma gramática que na folha anterior

$$I = \{ \; S' \to E \cdot \quad E \to E \cdot + T \; \}$$

$$goto(I, +) = closure(\{ E \to E + \cdot T \})$$

$$= \{ \; E \to E + \cdot T \quad T \to \cdot (E) \quad T \to \cdot id \; \}$$

Construção da coleção canónica de estados de items

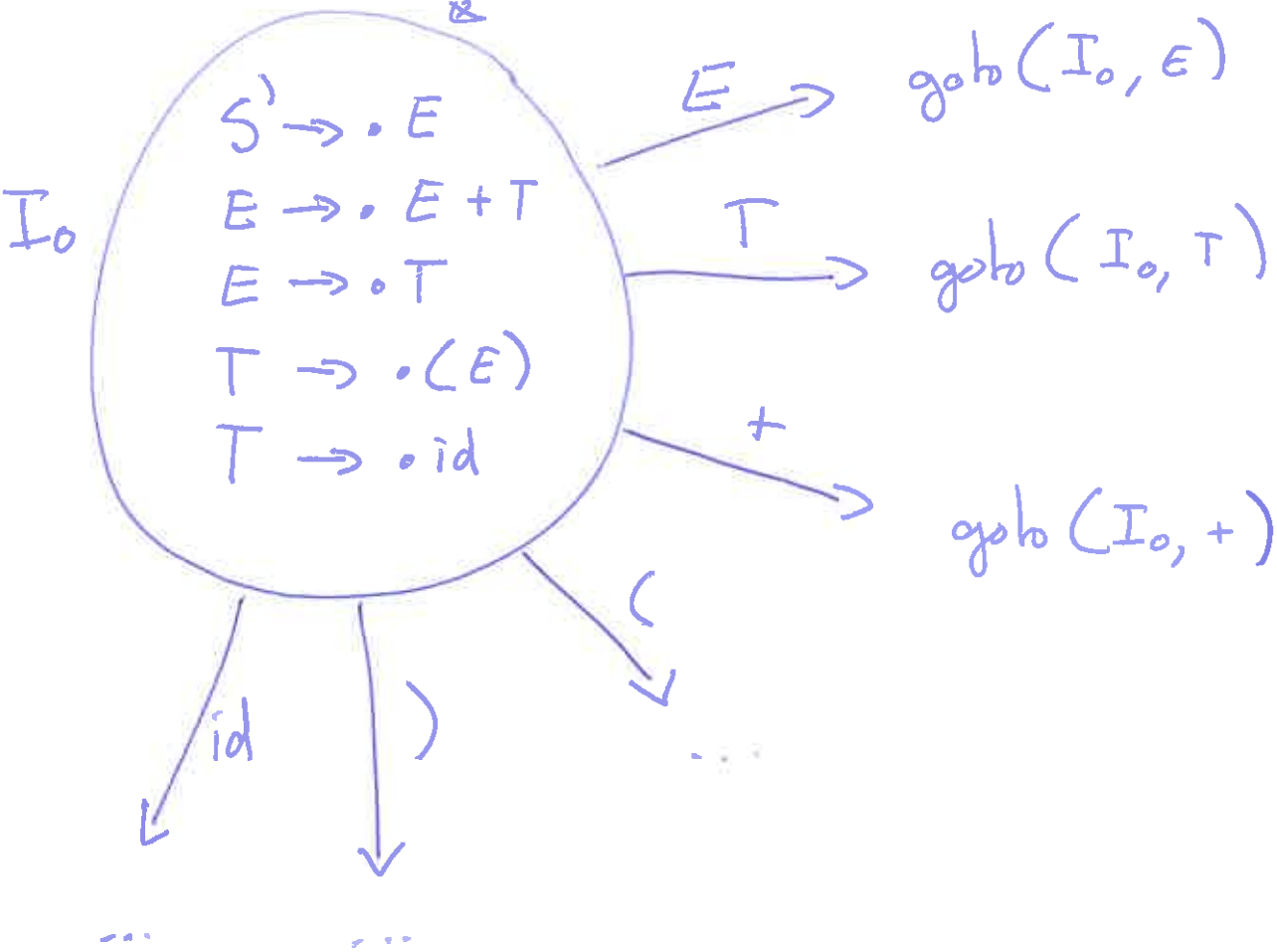
1) Acrescentar à gramática G um novo símbolo inicial S' e a regra S' → S (onde S é o símbolo inicial da gramática original G).
A nova gramática é G'.

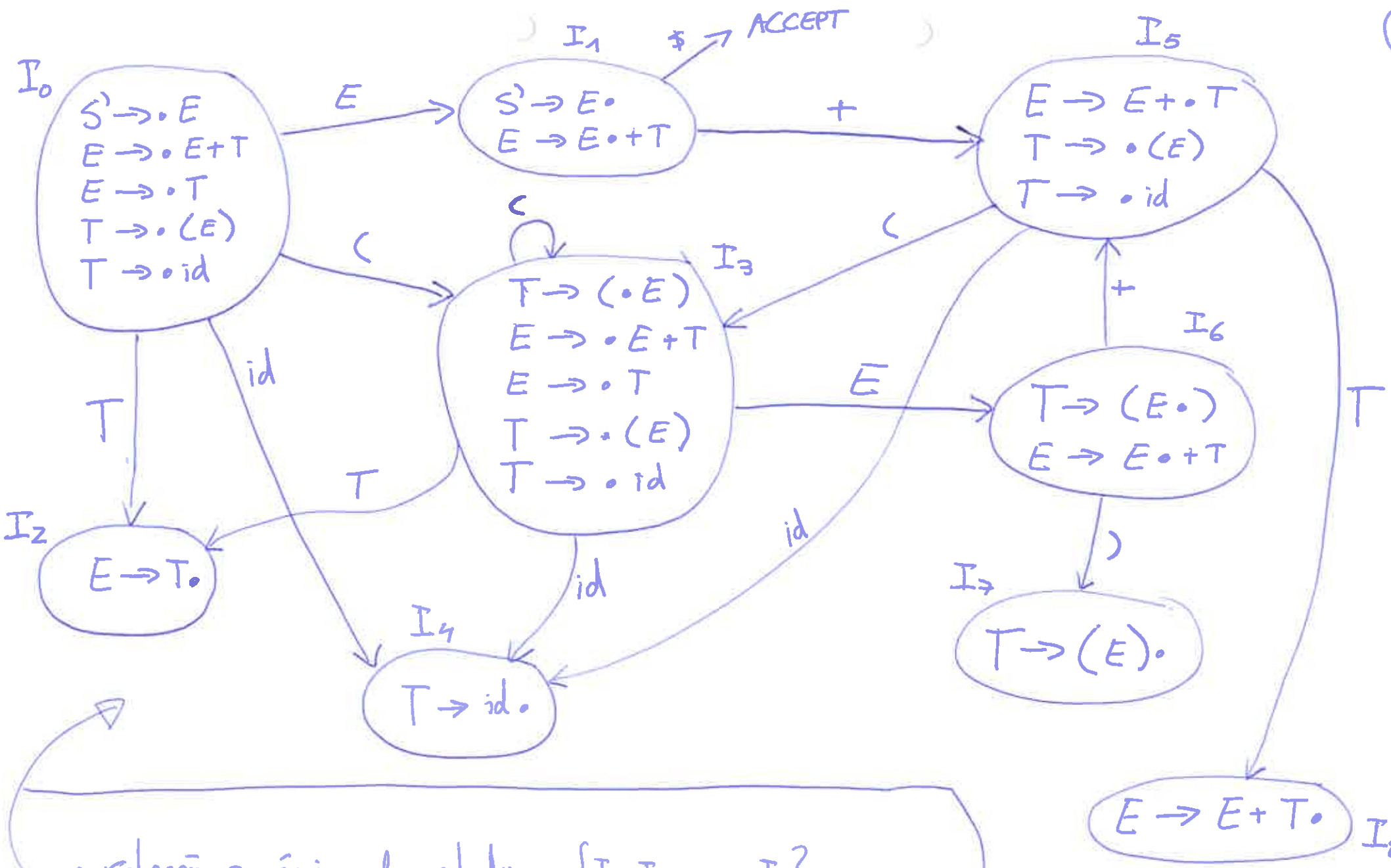
```
procedure items(G')
begin
    C := { closure({ S' → .S }) }
    repeat
        for each set of items I in C
            and each grammar symbol X
            such that goto(I, X) is not
            empty and not in C
        do
            add goto(I, X) to C
    until no more sets of items can be
        added to C
end
```

$S' \to E$
$E \to E + T$
$E \to T$
$T \to (E)$
$T \to id$

---

closure ({ $S' \to \cdot E$ })





- coleção canónica de estados = $\{I_0, I_1, \ldots, I_8\}$.
- Goto-graph ou transition diagram

Para obter as tabelas de parsing LR(0)

Input: gramática aumentada G'
Output: Tabelas action e goto para o parser LR(0)

Método

1) Construir $C = \{ I_0, I_1, \ldots I_n \}$, a coleção canónica de conjuntos de items

2) Cada I_i corresponde ao estado i.
Acções do parser para o estado i:

a) Se $A \to \alpha \cdot a\beta$ estiver em I_i e $goto(I_i, a) = I_j$, então Action$[i, a]$ é "Shift j"

a é terminal

b) Se $A \to \alpha \cdot$ estiver em I_i então Action$[i, a]$ é "Reduce $A \to \alpha$" para todos os inputs.

A ≠ S'

c) Se $S' \to S \cdot$ estiver em I_i então Action$[i, \$]$ é "Accept"

3) Se $goto(I_i, A) = I_j$
então $goto[i, A] = j$

4) Todas as entradas não definidas por 2) e 3) são "Error"

5) O estado inicial é aquele que é construído a partir do fecho de conjunto de itens de $S' \rightarrow .S$

A tabela de parsing na folha 6 foi construída deste modo.

Parser SLR(1) ⟶ simple LR

- praticamente idêntico ao que vimos (LR(0)).
  A única diferença é que não fazemos a redução qualquer que seja o próximo token.

O passo 2b) da folha 15 passa a ser:

> Se $A \to \alpha \cdot$ estiver em I_i
> então Action[i, a] é "Reduce $A \to \alpha$"
> apenas para os tokens $\in$ FOLLOW(A)

No exemplo dado

FOLLOW(S') = {$}
FOLLOW(E) = {$,), +}
FOLLOW(T) = {$,), +}

Ou seja, a tabela SLR(1) ficaria:

Estado no topo do Stack	Action					Goto	
	id	+	(	)	$	E	T
0	s4		s3			1	2
1		s5			Accept		
2		r2		r2	r2		
3	s4		s3			6	2
4		r4		r4	r4		
5	s4		s3				8
6		s5		s7			
7		r3		r3	r3		
8		r1		r1	r1		