Análise Sintática

Parte III – Análise Sintática Bottom-Up

Análise Bottom-UP

- A análise sintática bottom-up é conhecida como análise de **empilhar e reduzir**
- A análise gramatical de empilhar e reduzir tenta construir uma árvore gramatical para uma cadeia de entrada
  - Começa pelas folhas (cadeia) e vai em direção à raiz (símbolo inicial da gramática)
  - Podemos pensar neste processo como o de "reduzir" uma cadeia w ao símbolo de partida de uma gramática

Análise Bottom-UP

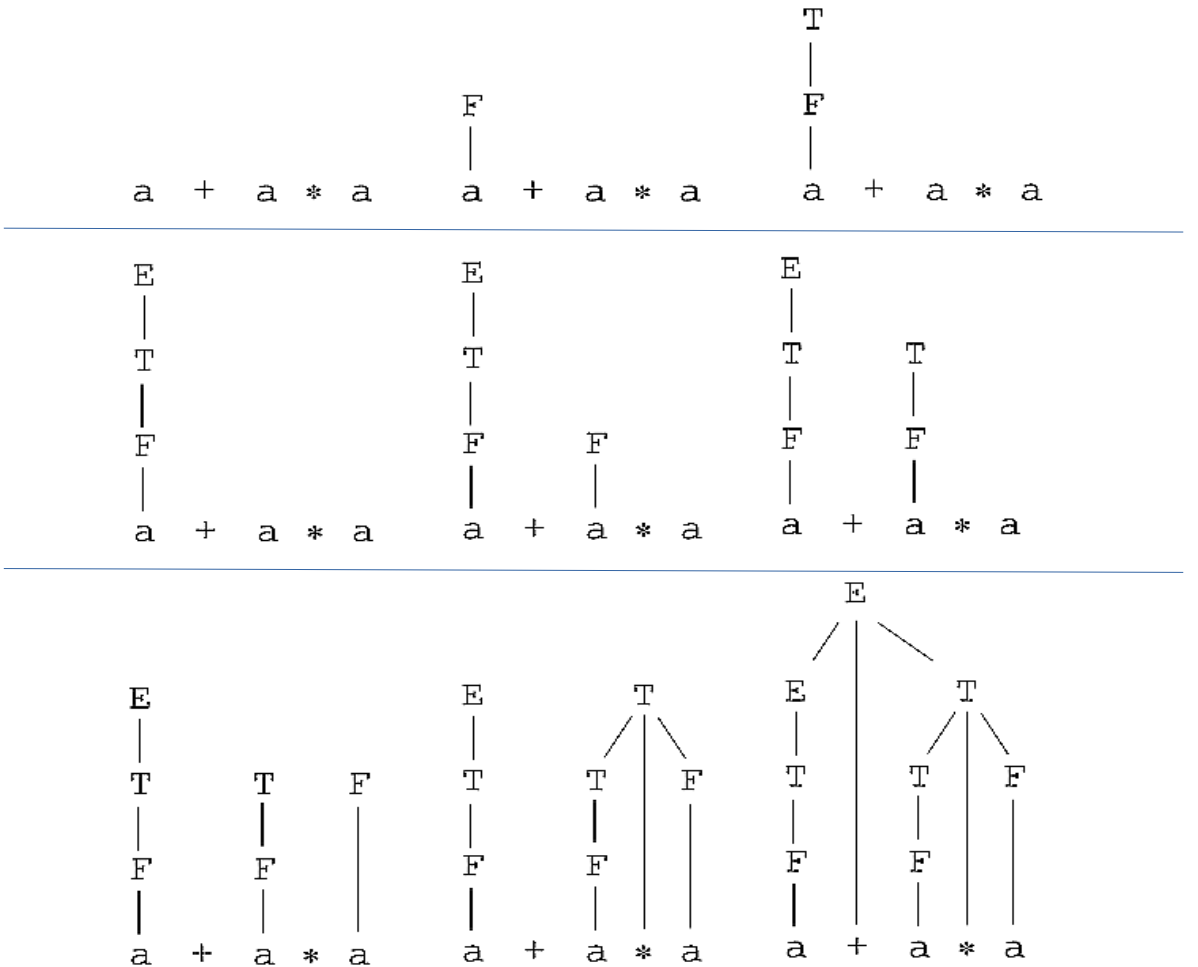
- A cada passo de **redução**, uma subcadeia particular, que reconheça o **lado direito** de uma produção, é substituída pelo símbolo à **esquerda** daquela produção
  - Se a subcadeia tiver sido escolhida corretamente a cada passo, uma **derivação mais à direita** terá sido rastreada na ordem inversa

Gramática:

1. E → E + T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → a

Sentença: a + a * a

Bottom-up:



a+a*a ⇐ F+a*a ⇐ T+a*a ⇐ E+a*a ⇐ E+F*a ⇐ E+T*a

Análise Bottom-UP

- A análise ascendente é mais complicada de implementar, tanto para um analisador escrito à mão quanto para geradores

- Mais geral que Top-Down
  - Impõe menos restrições à gramática
  - Por exemplo, recursão à esquerda e prefixos em comum não são problemas para as técnicas de análise ascendente

Reduções

- **Redução**: *substituição do lado direito de uma produção pelo não terminal correspondente (lado esquerdo)*
- A sequência de reduções da análise ascendente equivale a uma **derivação mais à direita**, lida de trás pra frente

Lembrando: Para uma gramática não ambígua, cada entrada só pode ter uma única derivação mais à direita

- *Isso quer dizer que a sequência de reduções também é única*

- O trabalho do analisador é então achar qual a próxima redução que tem que ser feita a cada passo
  - **Como saber quando fazer a redução?**

Handles (Prefixos Viáveis)

- **Handle**: sequência de símbolos do lado direito da produção, tais que suas reduções levam, no final, ao símbolo inicial da gramática
- Se a gramática não é ambígua então cada forma sentencial à direita tem um *handle único*
  - Veremos no **exemplo 2**
- Simplesmente procurar por regras com lados direitos que casem com o topo da pilha não é o suficiente
  - Veremos no **exemplo 3**

Handles (Prefixos Viáveis)

- **Ponto Fundamental**:
  - Como sabemos que achamos um *handle* sem primeiro gerar um monte de derivações?
- **Resposta**:
  - Usamos a informação do que já vimos na entrada, o "estado" do parser e a próxima palavra da entrada (*lookahead*)

- Técnicas para construir um AFD para *handles*
  - Veremos adiante...

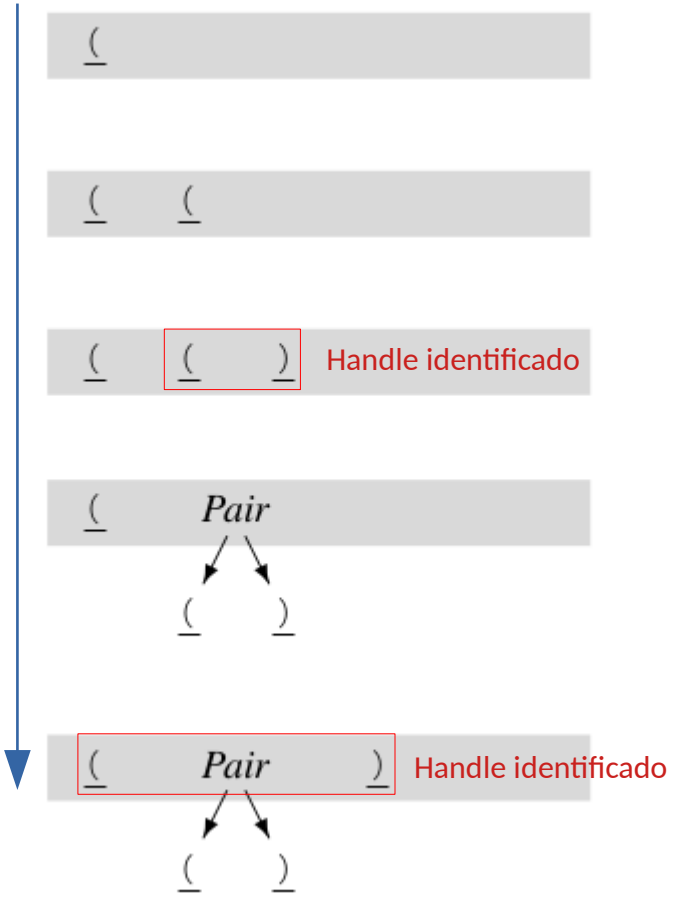
Exemplo 1

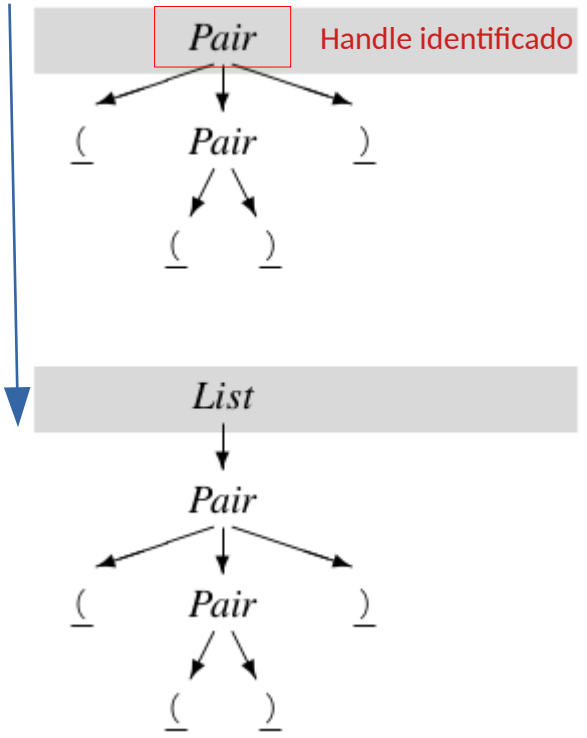
Gramática:

```
List → List Pair |
       Pair
Pair → ( Pair ) |
       (   )
```

Sentença:

"(( ))"





Exemplo: Torczon pg. 122

Exemplo 2

Gramática Ambígua:

E → E + E

E → E * E

E → id

Sentença:

"id + id * id"

Duas derivações possíveis!

Problemas com a escolha do *handle*!

<u>id</u> + id * id
E + <u>id</u> * id
E + E * <u>id</u>
E + <u>E * E</u>
<u>E + E</u>
E

Derivação 1

<u>id</u> + id * id
E + <u>id</u> * id
E + E * <u>id</u>
<u>E + E</u> * E
<u>E * E</u>
E

Derivação 2

Exemplo 3

Gramática

S → aABe

A → Abc | b

B → d

Sentença:

"abbcde"

abbcde

aAbcde

“b” nesse caso pode ser um handle?

Exemplo 3

Gramática:

S → aABe

A → Abc | b

B → d

Sentença:

"abbcde"

abbcde

aAbcde

aAAcde

aAAcBe

??

“b” nesse caso pode ser um handle?

Não... pois se substituir “b” por “A” não será mais possível chegar ao símbolo inicial da gramática

Exemplo 3

Gramática:

S → aABe

A → Abc | b

B → d

Sentença:

"abbcde"

abbcde

aAbcde

aAde

aABe

S

É necessário escolher o handle correto...

Nesse caso não reduzimos “b”, mas sim “Abc”

Funcionamento do Analisador Bottom-up

Estrutura de dados

- **Pilha:**
  - Armazena os símbolos gramaticais
  - Inicialmente contém apenas $

- **Buffer:**
  - Armazena a cadeia w a ser decomposta
  - Inicia com $w$$

*Usamos $ para marcar o fundo da pilha e o final da entrada (direita)

Funcionamento do Analisador Bottom-up

Ações

- As ações que podem ser realizadas por um reconhecedor bottom-up são as seguintes:
  - **Empilha (shift)** - coloca no topo da pilha o símbolo que está sendo lido e avança o cabeçote de leitura
  - **Reduz (reduce)** - substitui o handle do topo da pilha pelo não-terminal correspondente
  - **Aceita** - reconhece que a sentença de entrada foi gerada pela gramática
  - **Erro** - chama uma sub-rotina de atendimento a erros

Funcionamento do Analisador Bottom-up

Operação

- Empilha zero ou mais símbolos até que um handle H surja no topo da pilha
- Reduz o handle H para o lado esquerdo da produção apropriada
- Repete-se este ciclo até que:
  - Pilha contenha o símbolo de partida e entrada vazia
  - Ou erro detectado

Exemplo 1

Gramática

P → E

E → E + T

E → T

T → id (E)

T → id

Sentença:

"id (id + id)"

Pilha	Entrada	Ação
$	id (id + id) $	shift
$ id	(id + id) $	shift
$ id (	id + id) $	shift
$ id (id	+ id) $	reduce T → id
$ id (T	+ id) $	reduce T → E
$ id (E	+ id) $	shift
$ id (E +	id) $	shift
$ id (E + id	) $	reduce T → id
$ id (E + T	) $	reduce E → E + T
$ id (E	) $	shift
$ id (E)	$	reduce T → id(E)
$ T	$	reduce E → T
$ E	$	reduce P → E
$ P	$	accept

Exemplo 1

Gramática

P → E

E → E + T

E → T

T → id (E)

T → id

Sentença:

"id (id + id)"

Alguma observação neste exemplo??

Pilha	Entrada	Ação
$	id (id + id) $	shift
$ id	(id + id) $	shift
$ id (	id + id) $	shift
$ id (id	+ id) $	reduce T → id
$ id (T	+ id) $	reduce T → E
$ id (E	+ id) $	shift
$ id (E +	id) $	shift
$ id (E + id	) $	reduce T → id
$ id (E + T	) $	reduce E → E + T
$ id (E	) $	shift
$ id (E)	$	reduce T → id(E)
$ T	$	reduce E → T
$ E	$	reduce P → E
$ P	$	accept

Conflitos

- Podemos atingir uma configuração na qual não sabemos como proceder
  - Mesmo conhecendo o conteúdo da pilha
  - Conflitos!!

- **Conflito empilhar/reduzir**
- **Conflito reduzir/reduzir**

Conflitos

- **Conflito empilhar/reduzir**
  - Possibilidade de empilhar ou reduzir, de acordo com o estado do processamento
  - **Exemplo (anterior):**
    - Temos na pilha: $id
    - Entrada: "(id..."
    - O que fazer?
      - Reduz usando T → id, ou
      - Empilhar para reduzir usando T → id (E) mais tarde?

Conflitos

- **Conflito empilhar/reduzir**
  - **Exemplo 2:**

S → abc | a

Empilha ou reduz?

Pilha	Entrada	Ação
$	abc$	empilha
$a	bc$	???

Conflitos

- **Conflito empilhar/reduzir**
  - **Exemplo 2:**

S → abc | a

Pilha	Entrada	Ação
$	abc$	empilha
$a	bc$	empilha
$ab	c$	empilha
$abc	$	reduz
$S	$	aceita

Nesse caso, deve-se empilhar!

Conflitos

- **Conflito reduzir/reduzir**
  - Possibilidade de reduzir de múltiplas formas, de acordo com o estado do processamento
  - **Exemplo:**
    - Temos na pilha: $id
    - Gramática contém (por exemplo):

      S → id

      R → id
    - Reduzir usando qual produção?

Conflitos

- Infelizmente não há uma maneira para dizer qual ação devemos tomar ou como resolver conflitos empilha-reduz e reduz-reduz
- Para isso, devemos levar em consideração algumas informações adicionais
  - Mais símbolos a frente, por exemplo

- **Esses problemas geralmente são causados por gramáticas ambíguas!**

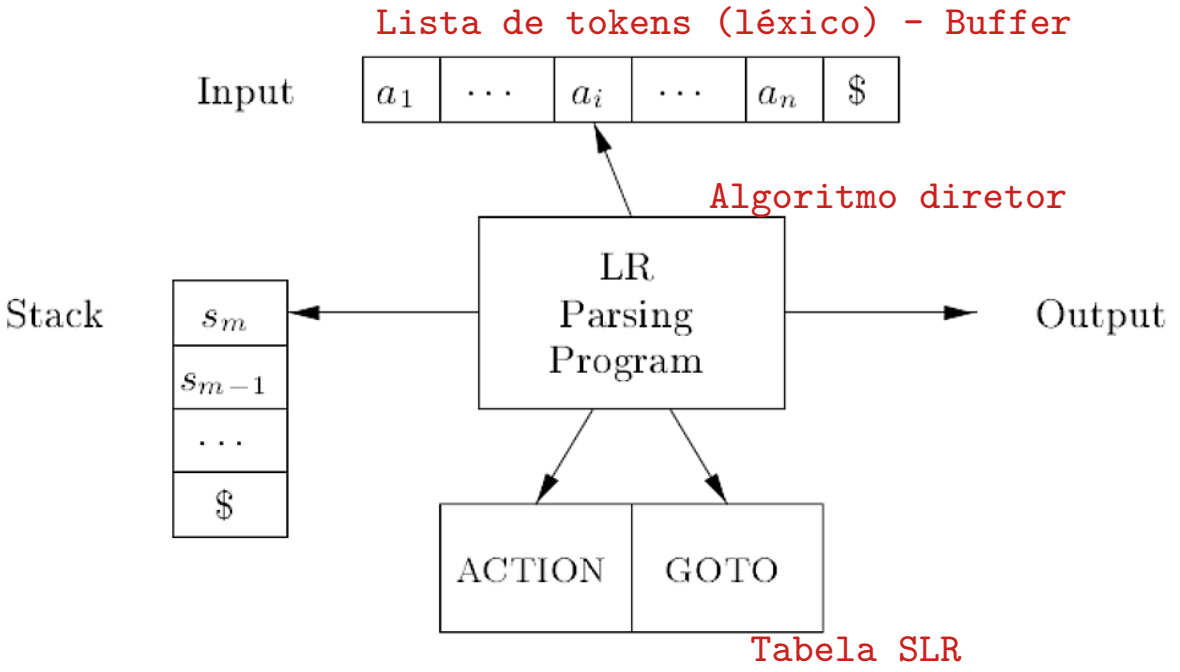
Implementações de Analisadores Bottom-Up

- Atualmente, o tipo mais prevalente de analisadores ascendentes é baseado em um conceito LR(k)
  - L: Leitura da esquerda para a direita (*left to right*)
  - R: Derivação mais a direita (*rightmost derivation*)
  - k: símbolos a frente lidos
- Vários algoritmos para o parsing empilha-reduz:
  - LR(0): Usa somente a pilha (k=0)
  - SLR(1): Simple LR, usa 1 token da entrada
  - LR(1)
  - LALR(1): Lookahead LR
  - Diferentes tabelas e linguagens que podem ser analisadas

SLR (Simple LR)

- A análise SLR é uma forma básica de análise LR na qual usamos o FOLLOW para resolver conflitos de empilhamento ou redução
- Em suma, nós usamos a redução $A \rightarrow \alpha$ apenas quando o próximo token na entrada estiver em FOLLOW(A)

SLR
Componentes



Construção da tabela SLR

- Conceitos que temos que entender para a construção da tabela:

  1) Itens
  2) Gramática aumentada
  3) Operação de fechamento
  4) Operação de desvio
  5) Construção do **conjunto de itens**

     Deve-se calcular os conjuntos canônicos de itens, ou seja, todos os itens alcançáveis a partir de um conjunto de regras da gramática.
     Isso dá origem a um autômato.

Construção da tabela SLR

1) Itens LR (ou simplesmente itens):

- Um item para uma gramática G é uma produção de G com um ponto (•) em alguma de suas posições no lado direito
- Exemplo:
  - Para A → CDE, temos 4 itens possíveis:
    A → •CDE
    A → C•DE
    A → CD•E
    A → CDE•

OBS: A → ε gera apenas A → •

A posição do • serve para indicar o que já foi analisado.

Itens com a marca no final são itens de redução (ex: A → CDE•).

Construção da tabela SLR

2) Gramática aumentada:

- Se G é uma gramática com símbolo inicial S, então G' (gramática aumentada) para G é G com um novo símbolo de partida S', mais a produção S'→ S
- Serve para indicar ao analisador quando deve parar a análise e indicar se houve a aceitação da entrada
  - Isso ocorre quando houver a redução de S para S'

Exemplo:

G:

```
List → List Pair
       | Pair
Pair → ( Pair )
       | ( )
```

G':

```
List' → List
List → List Pair
       | Pair
Pair → ( Pair )
       | ( )
```

Construção da tabela SLR

3) Operação de Fechamento

- Considere I como o conjunto de itens para G
- O *fechamento(I)* é o conjunto de itens construídos a partir de I
- **Regras**:
  1) Cada item em I é adicionado ao fechamento(I)
  2) Se A → a•Bb estiver em fechamento(I) e B → c for uma produção, adicionar o item B → •c ao conjunto I
     - Repete-se até que não se possa mais adicionar novos itens

Construção da tabela SLR

Exemplo:

Dada a gramática:

E'→ E

E → E + T | T

T → T*F | F

F → (E) | id

Se I for o conjunto de um item {**[E' → •E]**}, então

fechamento(I) = {

E' → •E *{ponto de partida - R1}*

E → •E + T

E → •T

T → •T*F

T → •F

F → •(E)

F → •id

}

Construção da tabela SLR

4) Operação de Desvio

- Desvio(I,X)
  - Conjunto de itens I e um símbolo X
  - Retorna um conjunto de itens
- Cálculo do desvio a partir do estado I ao ler X:
  - Mover ponto para direita em todos os itens de I onde o ponto precede X
    - Para todas as regras $A \rightarrow \alpha \bullet X \beta$ em C, retorna $A \rightarrow \alpha X \bullet \beta$
  - Calcular o fechamento deste conjunto de itens

Construção da tabela SLR

Exemplo:

Dada a gramática:

S' → T

T → F | T * F

F → id | (T)

e **I=**

{ S' → •T

T → •F

T → •T * F

F → •id

F → •(T)}

Desvio (I,(=

F → (• T)

T → •F

T → •T * F

F → •id

F → •(T)

Desvio (I,id)=

F → id•

Construção da tabela SLR

Exemplo:

Dada a gramática:

S' → T

T → F | T * F

F → id | (T)

e I=

{ S' → •T

  T → •F

  T → •T * F

  F → •id

  F → •(T)}

Desvio (I,F)=

{

  T → F•

}

Desvio (I,T)=

{

  S' → T•

  T → T• * F

}

Construção da tabela SLR

5) Construção dos conjuntos

- A construção consiste em calcular todos os conjuntos a partir dos desvios
- Parte-se do fechamento(I_0), com I_0=[S'→ •S]
  - S' é o símbolo inicial da gramática aumentada
- Novos estados I_n são gerados

Construção da tabela SLR

5) Construção dos conjuntos de itens

```
proc itens(G')
{
   C = fechamento([[S'→ •S]])  {C é o conjunto de itens}
   repetir
      para cada conjunto de itens I em C e cada símbolo gramatical
      X tal que desvio(I,X) não seja vazio e não esteja em C
         incluir desvio(I,X) a C
   até que não haja mais conjuntos de itens a serem incluídos a C
}
```

Construção da tabela SLR

Exemplo:

Considere a gramática:

0) S'→ T

1) T → F

2) T → T * F

3) F → id

4) F → (T)

Estado inicial I_0

Fechamento de S'→ •T

I_0: S'→ •T

T → •F

T → •T * F

F → •id

F → •(T)

I_0:

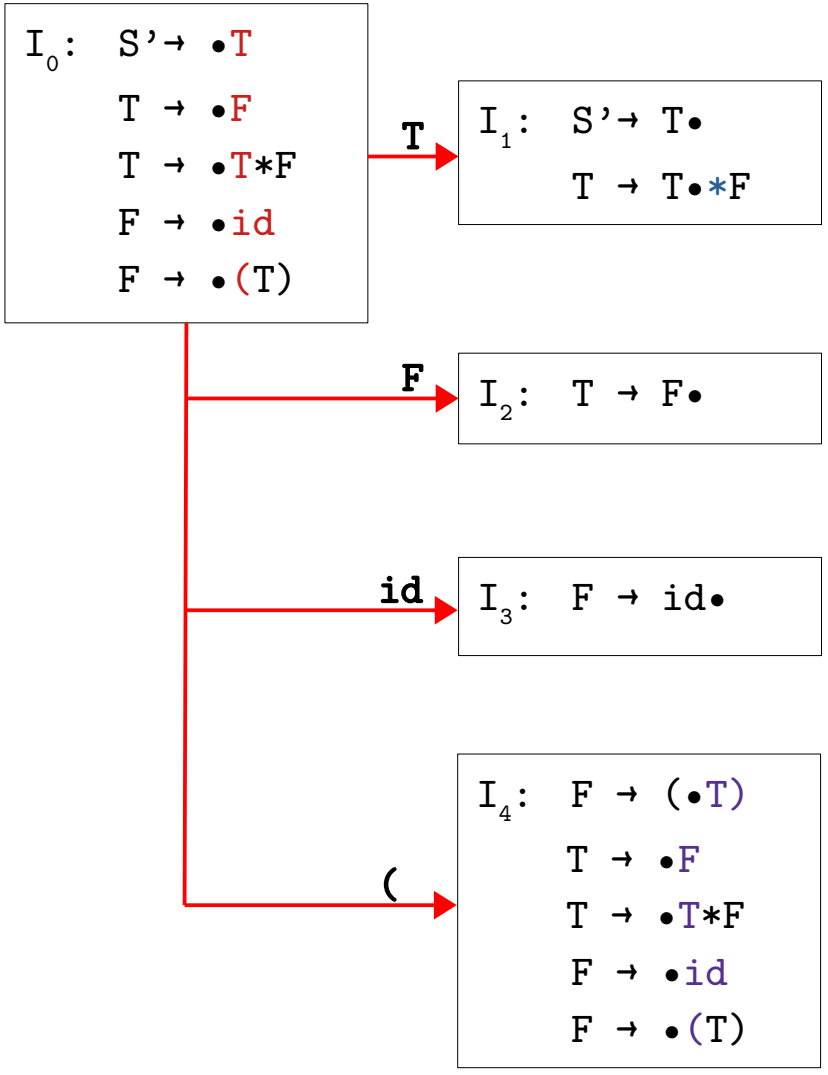
S' → •T

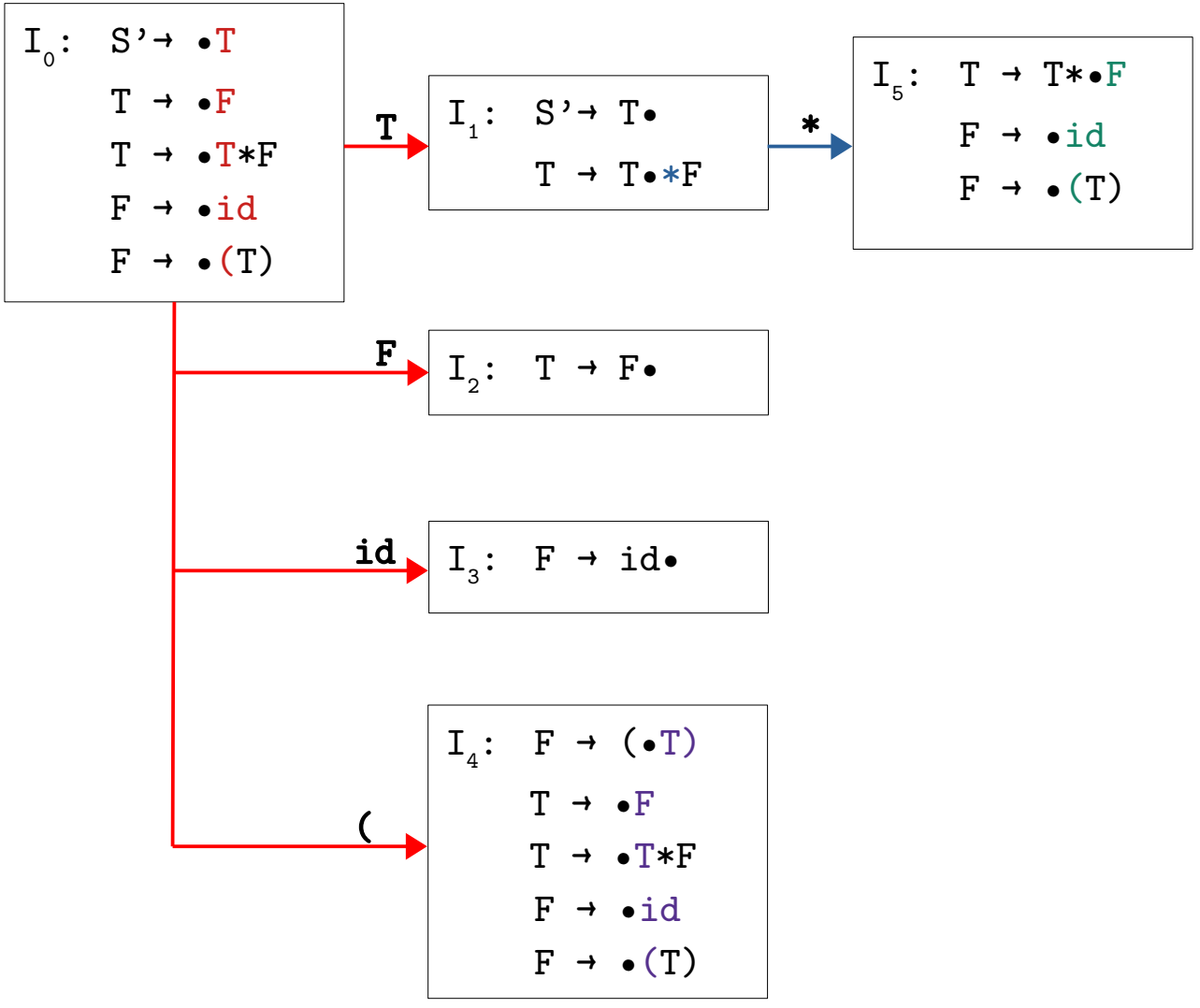
T → •F

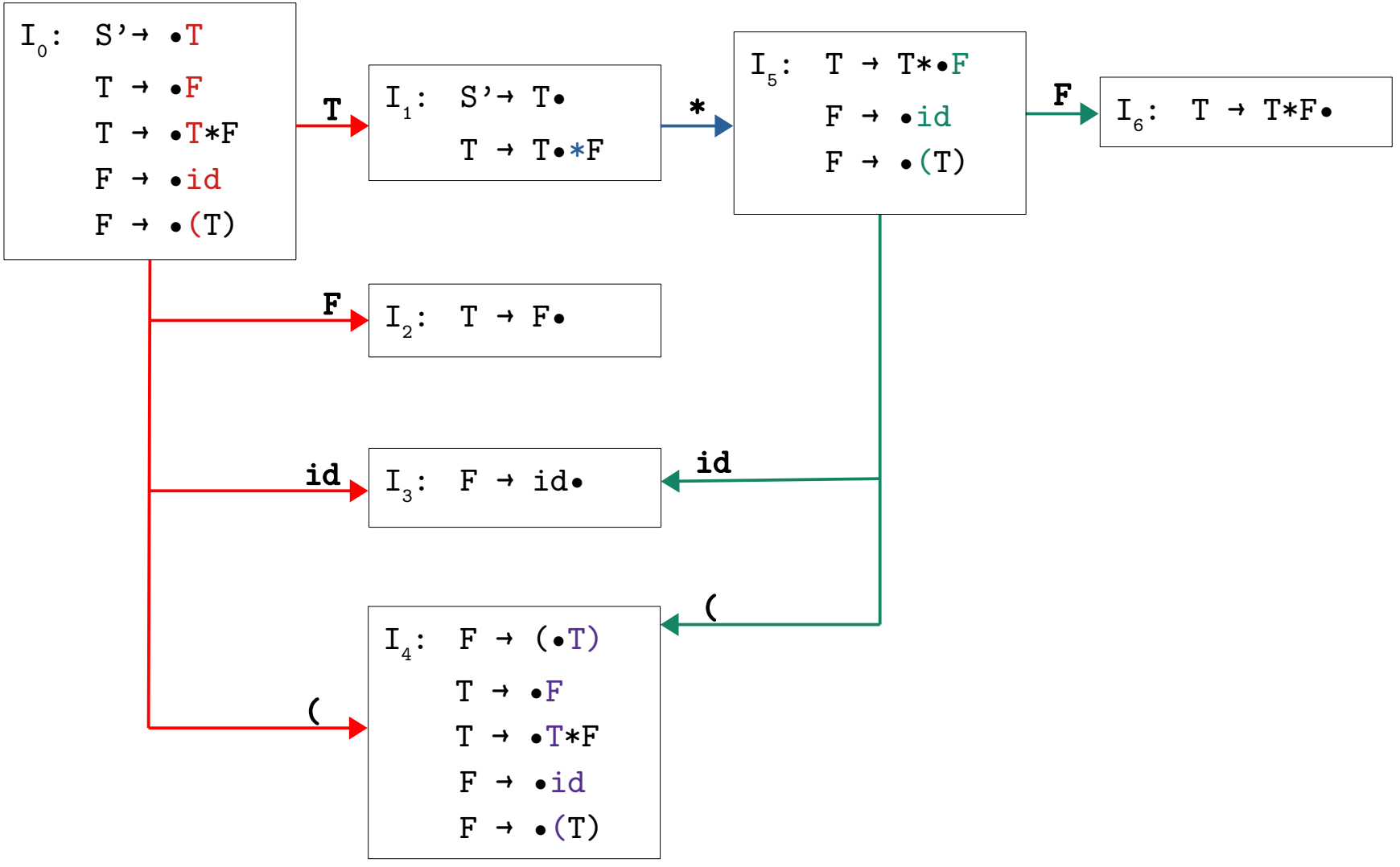
T → •T*F

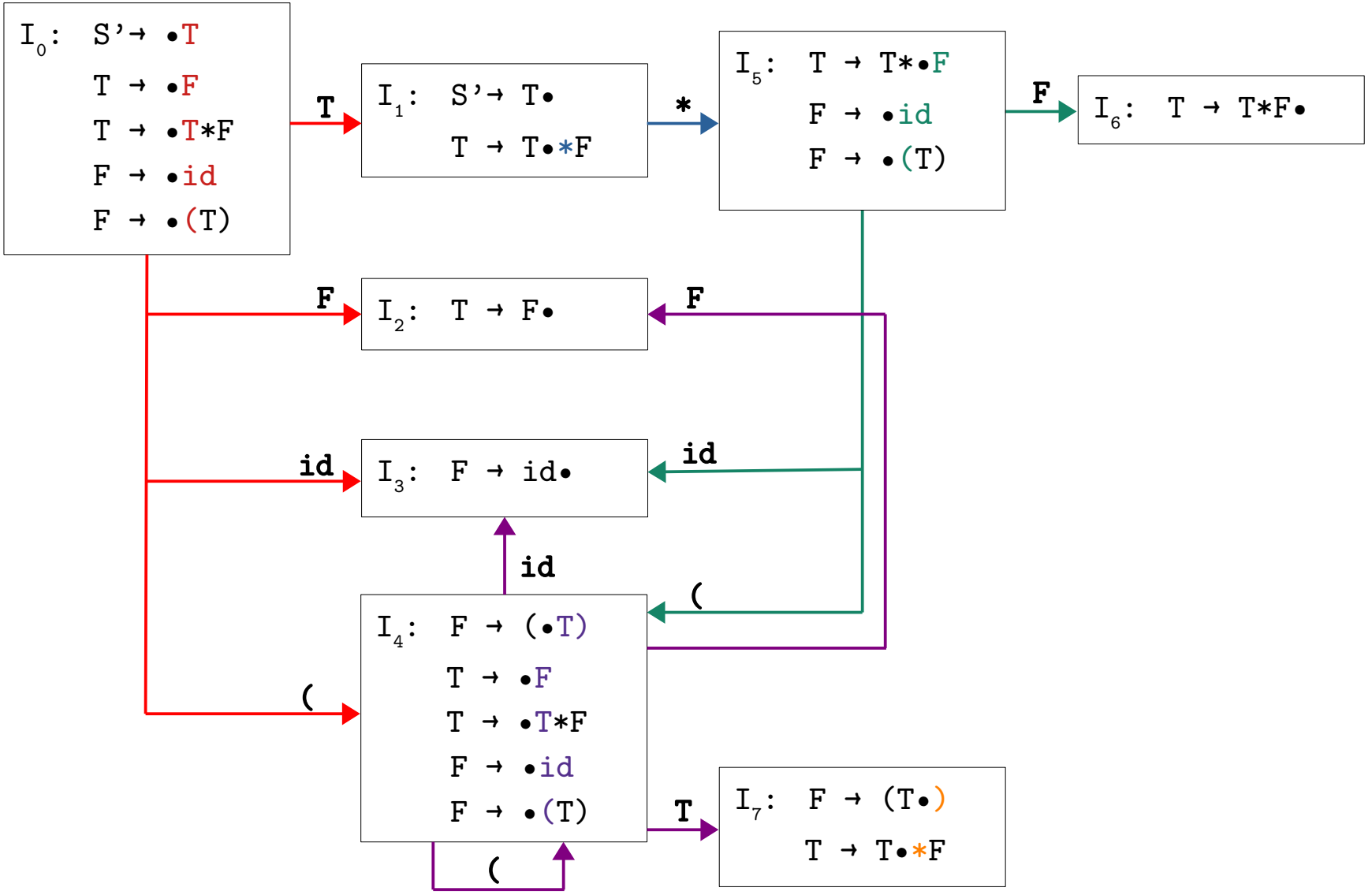
F → •id

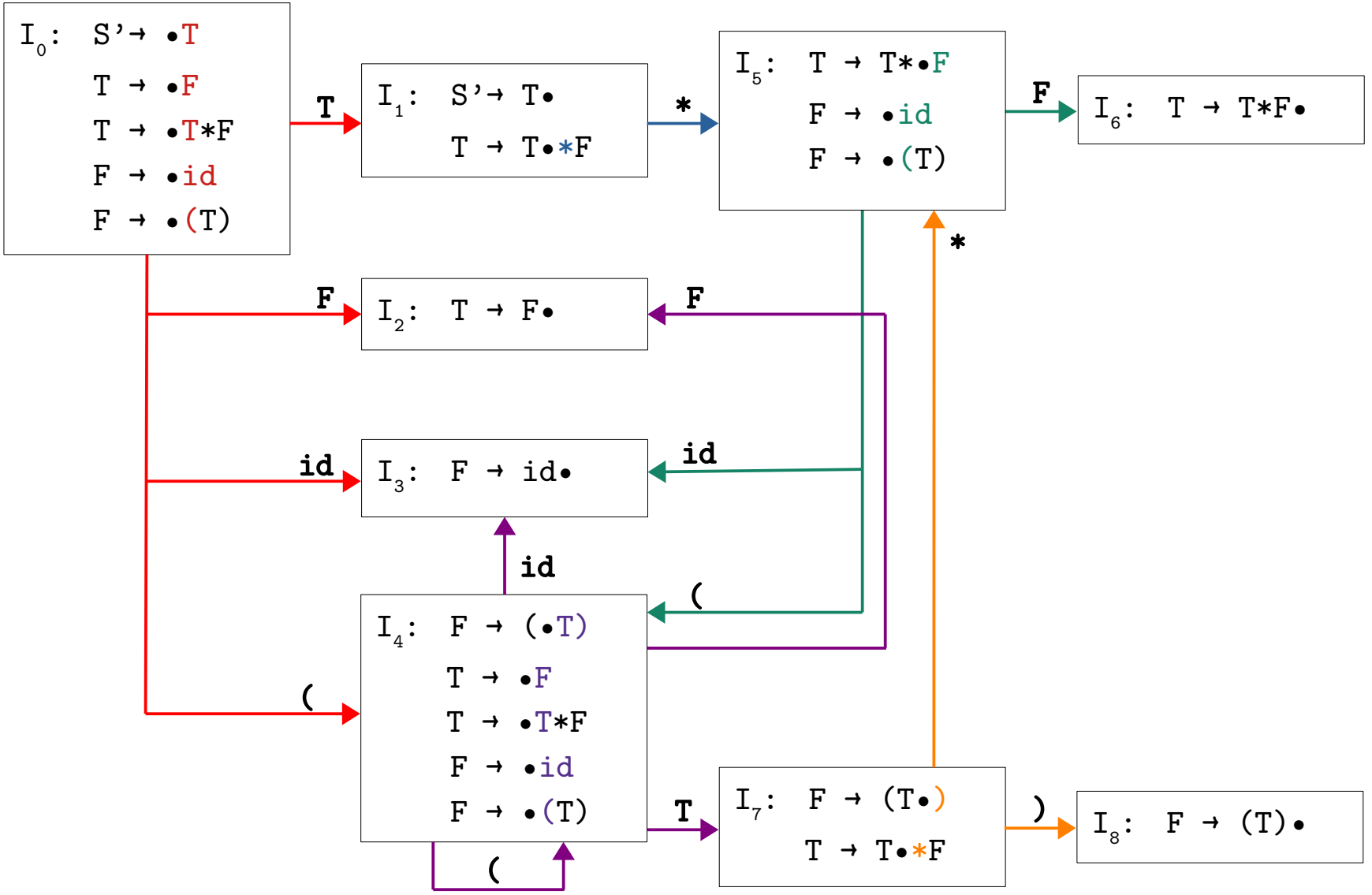
F → •(T)


I0: S'→ •T
T → •F
T → •T*F
F → •id
F → •(T)
T
I1: S'→ T•
T → T•*F
F
I2: T → F•
id
I3: F → id•
(
I4: F → (•T)
T → •F
T → •T*F
F → •id
F → •(T)


I0: S'→ •T
T → •F
T → •T*F
F → •id
F → •(T)
T
I1: S'→ T•
T → T•*F
*
I5: T → T*•F
F → •id
F → •(T)
F
I2: T → F•
id
I3: F → id•
(
I4: F → (•T)
T → •F
T → •T*F
F → •id
F → •(T)


I0: S'→ •T
T → •F
T → •T*F
F → •id
F → •(T)
T
I1: S'→ T•
T → T•*F
*
I5: T → T*•F
F → •id
F → •(T)
F
I6: T → T*F•
F
I2: T → F•
id
I3: F → id•
id
(
I4: F → (•T)
T → •F
T → •T*F
F → •id
F → •(T)
(


I0: S'→ •T
T → •F
T → •T*F
F → •id
F → •(T)
T
I1: S'→ T•
T → T•*F
*
I5: T → T*•F
F → •id
F → •(T)
F
I6: T → T*F•
F
I2: T → F•
F
id
I3: F → id•
id
id
(
I4: F → (•T)
T → •F
T → •T*F
F → •id
F → •(T)
(
T
I7: F → (T•)
T → T•*F
(


I0: S'→ •T
T → •F
T → •T*F
F → •id
F → •(T)
T
I1: S'→ T•
T → T•*F
*
I5: T → T*•F
F → •id
F → •(T)
F
I6: T → T*F•
*
F
I2: T → F•
F
id
I3: F → id•
id
id
(
I4: F → (•T)
T → •F
T → •T*F
F → •id
F → •(T)
(
(
T
I7: F → (T•)
T → T•*F
)
I8: F → (T)•

Construção da tabela SLR

Algoritmo

- Construção das funções de ação e desvio a partir do autômato que reconhece os prefixos viáveis (handles)

- Pré-requisito:
  - Cálculo do *Follow(A)* para cada não-terminal A da gramática

Construção da tabela SLR

Algoritmo

Entrada: Gramática aumentada G'

Saída: Funções sintáticas SLR ação e desvio para G'

Proc tabela(G')

{

1. Construir C={I_0..I_n} *//coleção de itens para G'*
2. O estado i é construído a partir de I_i. As ações sintáticas para o estado i são determinadas como segue:

   a) Se [A → α•aβ] estiver em I_i e desvio(I_i,a)=I_j, então estabelecer ação[i,a] em "empilha j". **Aqui a é terminal**. //R1

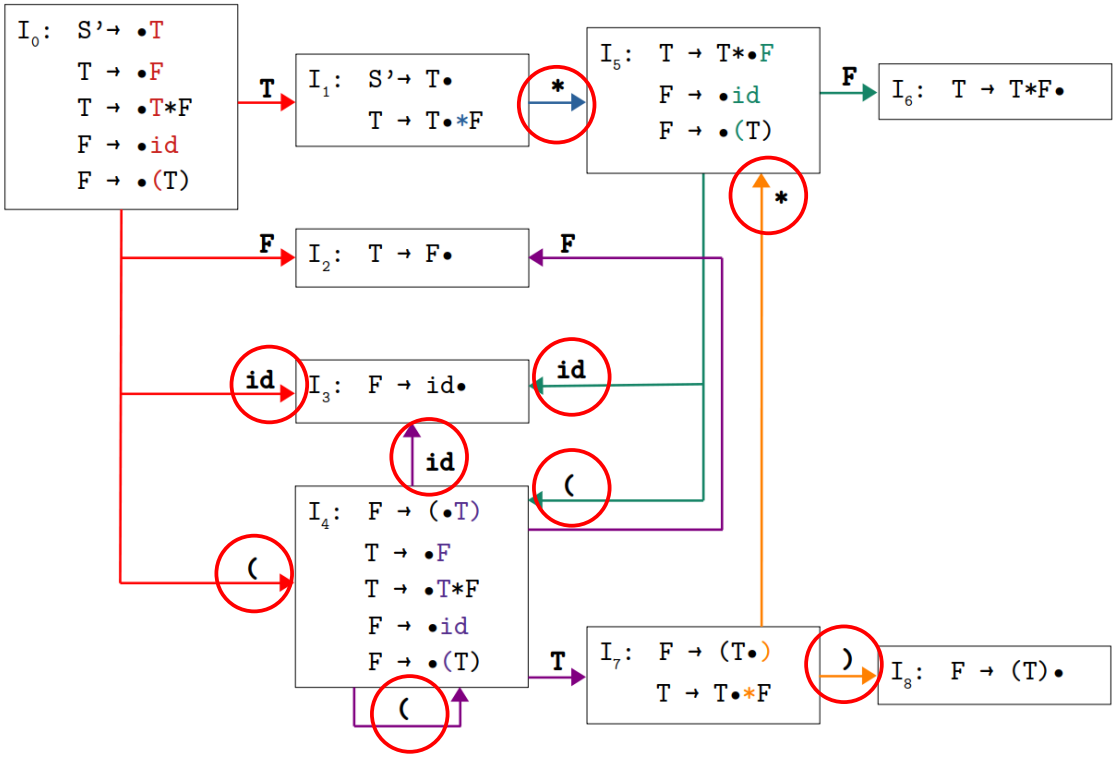
   b)Se [A → α•] estiver em I_i, então estabelecer ação[i,a] em "reduzir através de A→ α" para todo a em Follow(A). **Aqui A é diferente de S'**. //R2

   c)Se[S' → S•] estiver em I_i, então estabelecer ação[i,$] igual a "aceitar". //R3
3. As transições de desvio para o estado i são construídos pra todos os não-terminais A usando a regra: se desvio(I_i,A)=I_j, então desvio[i,A]=j //R4
4. Entradas não definidas: erro

}

Construção da tabela SLR

Algoritmo



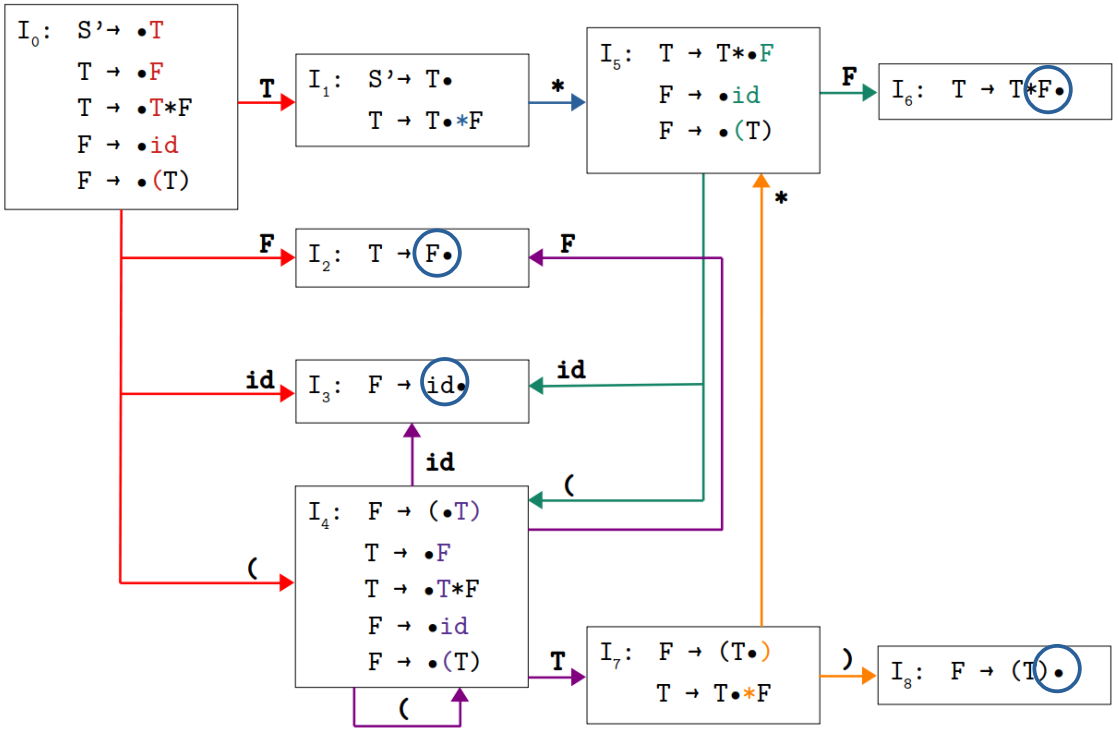
Aplicação de R1:

	*	(	)	id	$	T	F
0		E4		E3			
1	E5						
2							
3							
4		E4		E3			
5		E4		E3			
6							
7	E5		E8				
8							

Construção da tabela SLR

Algoritmo

0) S'→ T
1) T → F
2) T → T * F
3) F → id
4) F → (T)

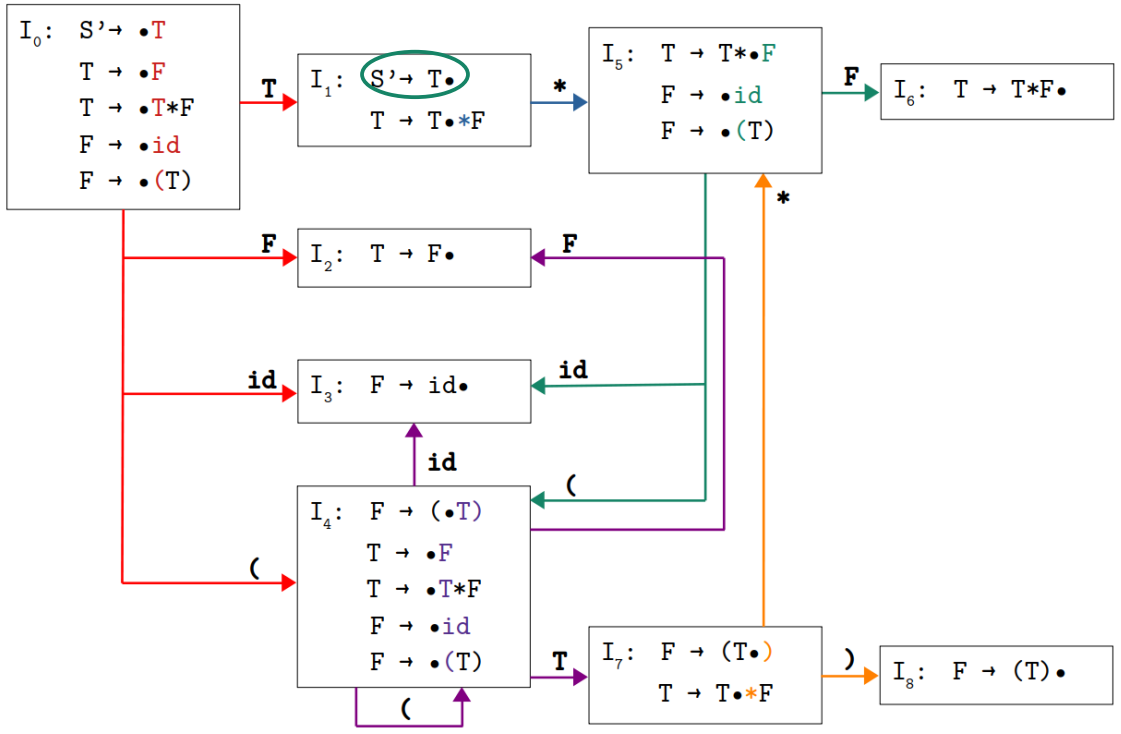


Aplicação de R2:

	*	(	)	id	$	T	F
0		E4		E3			
1	E5						
2	R1		R1		R1		
3	R3		R3		R3		
4		E4		E3			
5		E4		E3			
6	R2		R2		R2		
7	E5		E8				
8	R4		R4		R4		

Construção da tabela SLR

Algoritmo

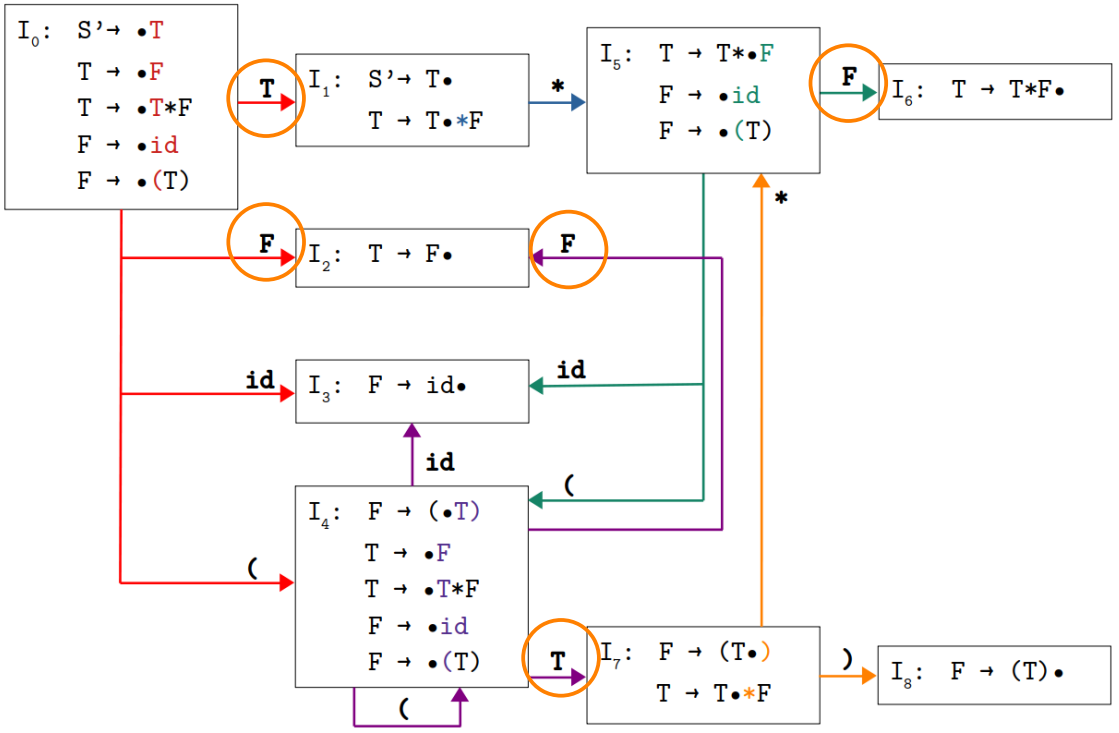


Aplicação de R3:

	*	(	)	id	$	T	F
0		E4		E3			
1	E5				AC		
2	R1		R1		R1		
3	R3		R3		R3		
4		E4		E3			
5		E4		E3			
6	R2		R2		R2		
7	E5		E8				
8	R4		R4		R4		

Construção da tabela SLR

Algoritmo



Aplicação de R4:

	*	(	)	id	$	T	F
0		E4		E3		1	2
1	E5				AC		
2	R1		R1		R1		
3	R3		R3		R3		
4		E4		E3		7	2
5		E4		E3			6
6	R2		R2		R2		
7	E5		E8				
8	R4		R4		R4		

Algoritmo de Análise LR

```
Entrada: sentença w e tabela com as funções ação e desvio para G
Proc Processa_LR(w)
{
    fazer ip apontar para o primeiro símbolo de w$;
    repetir
        seja s o estado do topo da pilha e a o símbolo apontado por ip;
        se ação[s,a] = empilhar s'
            empilhar a e em seguida s' no topo da pilha;
            avançar ip para o próximo símbolo de entrada;
        senão se ação[s,a] = reduzir A → β
            desempilhar 2*|β|;
            seja s' o estado agora no topo da pilha;
            empilhar A e em seguida desvio[s',A];
        senão se ação[s,a] = aceitar
            retorna "aceita";
        senão
            erro();
}
```

Algoritmo de Análise LR

0) S'→ T
1) T → F
2) T → T * F
3) F → id
4) F → (T)

	*	(	)	id	$	T	F
0		E4		E3		1	2
1	E5				AC		
2	R1		R1		R1		
3	R3		R3		R3		
4		E4		E3		7	2
5		E4		E3			6
6	R2		R2		R2		
7	E5		E8				
8	R4		R4		R4		

Exemplo: processamento para a entrada (id)*id.

Pilha	Entrada	Ação
0	(id)*id$	E4
0 (4	id)*id$	E3
0 (4 id 3	)*id$	R3
0 (4 F 2	)*id$	R1
0 (4 T 7	)*id$	E8
0 (4 T 7) 8	*id$	R4
0 F 2	*id$	R1
0 T 1	*id$	E5
0 T 1 * 5	id$	E3
0 T 1 * 5 id 3	$	R3
0 T 1 * 5 F 6	$	R2
0 T 1	$	**AC**

Tratamento de Erros

- Um analisador LR irá detectar um erro ao consultar a tabela de ações sintáticas e encontrar uma entrada de erro
  - Posição na matriz vazia
- Erro anunciado caso não haja continuação válida
  - Relatam o problema e param
- Esse comportamento evita que o compilador perca tempo tentando traduzir um programa incorreto
  - Encontra no máximo um erro de sintaxe por compilação

Tratamento de Erros

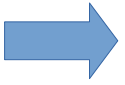
- Um analisador deve encontrar tantos erros de sintaxe quanto possível em cada compilação
- Permitir que o analisador se recupere de um erro movendo-se para um estado em que possa continuar a análise

Tratamento de Erros

- **Modo pânico:**
  - Linhas que contém reduções:
    - Células em branco são preenchidas com reduções
  - Demais células
    - Chamadas a rotinas de tratamento de erros

Tratamento de Erros

	*	(	)	id	$	T	F
0		E4		E3		1	2
1	E5				AC		
2	R1		R1		R1		
3	R3		R3		R3		
4		E4		E3		7	2
5		E4		E3			6
6	R2		R2		R2		
7	E5		E8				
8	R4		R4		R4		



	*	(	)	id	$	T	F
0	erro	E4	erro	E3	erro	1	2
1	E5	erro	erro	erro	AC		
2	R1	R1	R1	R1	R1		
3	R3	R3	R3	R3	R3		
4	erro	E4	erro	E3	erro	7	2
5	erro	E4	erro	E3	erro		6
6	R2	R2	R2	R2	R2		
7	E5	erro	E8	erro	erro		
8	R4	R4	R4	R4	R4		

Cada posição da matriz pode gerar um erro diferente e exigir um tratamento distinto. Isso vai depender da gramática.

Algoritmo de Análise LR

0) S'→ T
1) T → F
2) T → T * F
3) F → id
4) F → (T)

	*	(	)	id	$	T	F
0	erro	E4	erro	E3	erro	1	2
1	E5	erro	erro	erro	AC		
2	R1	R1	R1	R1	R1		
3	R3	R3	R3	R3	R3		
4	erro	E4	erro	E3	erro	7	2
5	erro	E4	erro	E3	erro		6
6	R2	R2	R2	R2	R2		
7	E5	erro	E8	erro	erro		
8	R4	R4	R4	R4	R4		

Exemplo1: processamento para a entrada (id*id.

Pilha	Entrada	Ação
$ 0	(id*id$	E4
$ 0 (4	id*id$	E3
$ 0 (4 id 3	*id$	R3
$ 0 (4 F 2	*id$	R1
$ 0 (4 T 7	*id$	E5
$ 0 (4 T 7 * 5	id$	E3
$ 0 (4 T 7 * 5 id 3	$	R3
$ 0 (4 T 7 * 5 F 6	$	R2
$ 0 (4 T 7	$	Erro[1]
$ 0 (4 T 7) 8	$	R4
$ 0 F 2	$	R1
$ 0 T 1	$	**Aceita (!)**

Erro[1] - *espera por um '*' ou ')'* → nesse caso pode empilhar um ')' e 8 (estado de 7 com ')') imaginário para continuar

Algoritmo de Análise LR

0) S'→ T
1) T → F
2) T → T * F
3) F → id
4) F → (T)

	*	(	)	id	$	T	F
0	erro	E4	erro	E3	erro	1	2
1	E5	erro	erro	erro	AC		
2	R1	R1	R1	R1	R1		
3	R3	R3	R3	R3	R3		
4	erro	E4	erro	E3	erro	7	2
5	erro	E4	erro	E3	erro		6
6	R2	R2	R2	R2	R2		
7	E5	erro	E8	erro	erro		
8	R4	R4	R4	R4	R4		

Erro[1] - *')' não esperado* → remover o ')'
Erro[2] - *'*' não esperado* → remover o '*'

Exemplo2: processamento para a entrada id)**id.

Pilha	Entrada	Ação
$ 0	id)**id$	E3
$ 0 id 3	)**id$	R3
$ 0 F 2	)**id$	R1
$ 0 T 1	)**id$	Erro[1]
$ 0 T 1	**id$	E5
$ 0 T 1 * 5	*id$	Erro[2]
$ 0 T 1 * 5	id$	E3
$ 0 T 1 * 5 id 3	$	R3
$ 0 T 1 * 5 F 6	$	R2
$ 0 T 1	$	**Aceita(!)**

Big Example

Gramática:

S → a

S → [L]

L → L;S

L → S

Big Example

Passo 1: Gramática aumentada

Gramática:

0) S'→ S

1) S → a

2) S → [L]

3) L → L;S

4) L → S

Big Example

Passo 2: Construção do Conjunto de itens

Gramática:

0) S’→ S
1) S → a
2) S → [L]
3) L → L;S
4) L → S

Ponto de partida:

[S’ → •S]

Estado Inicial:

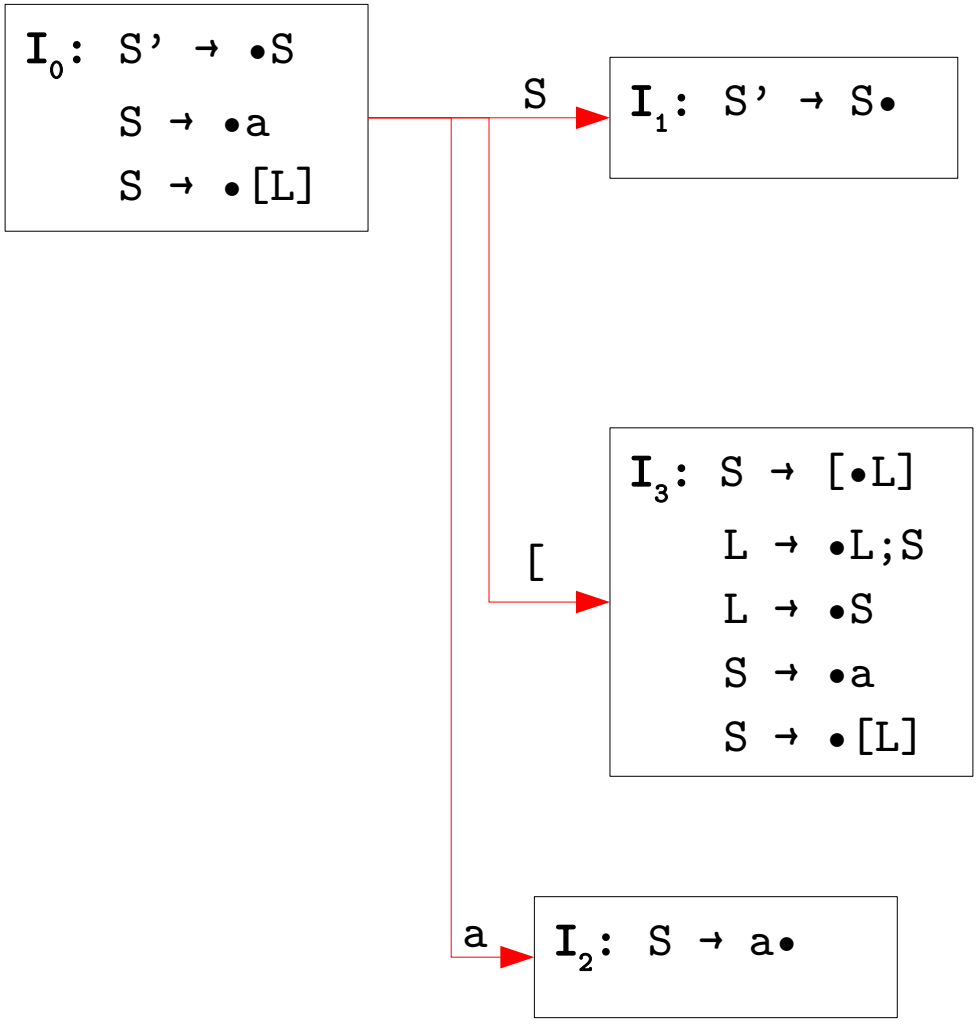
Fechamento de [S’ → •S]

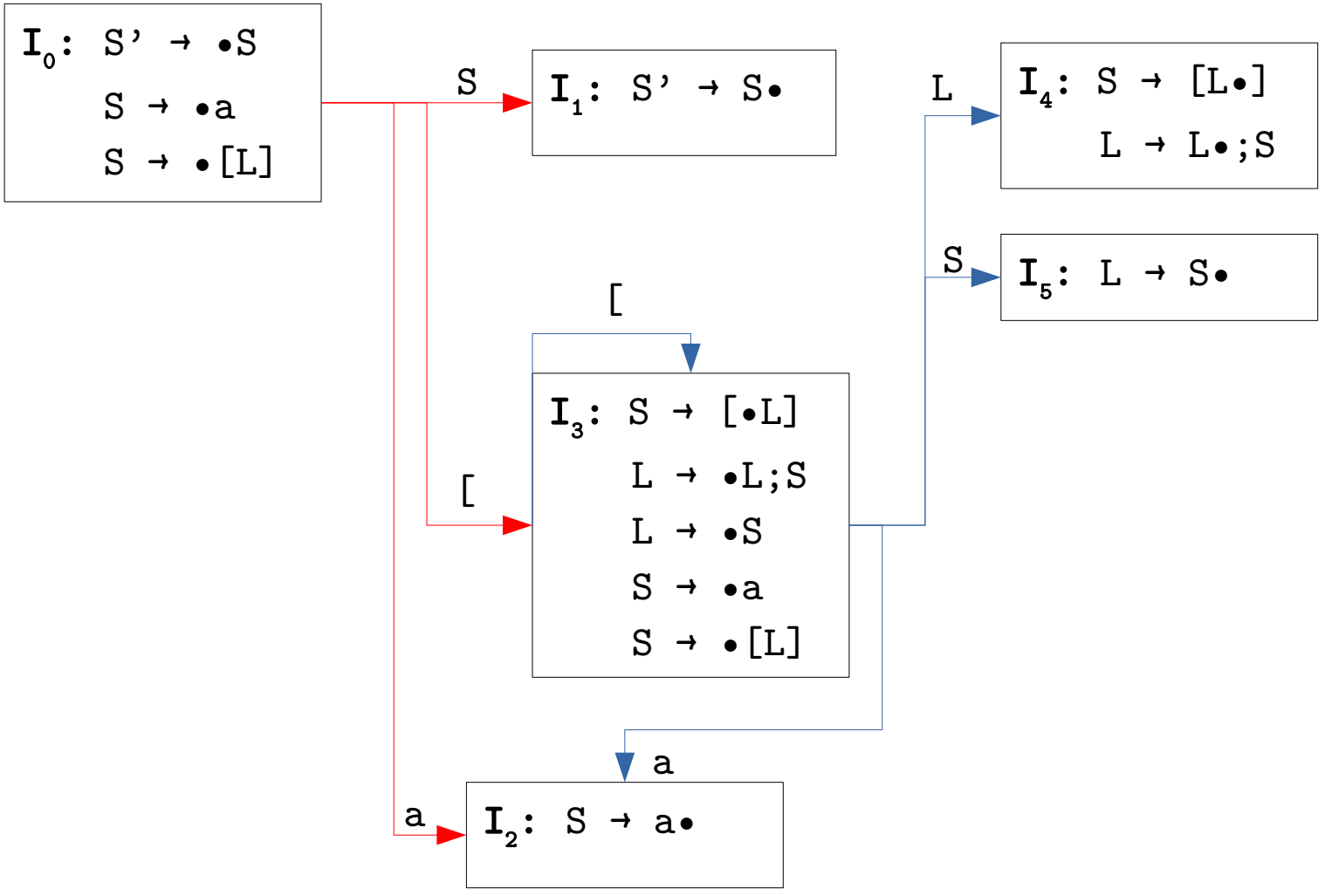
I_0:	S’ → •S
	S → •a
	S → •[L]

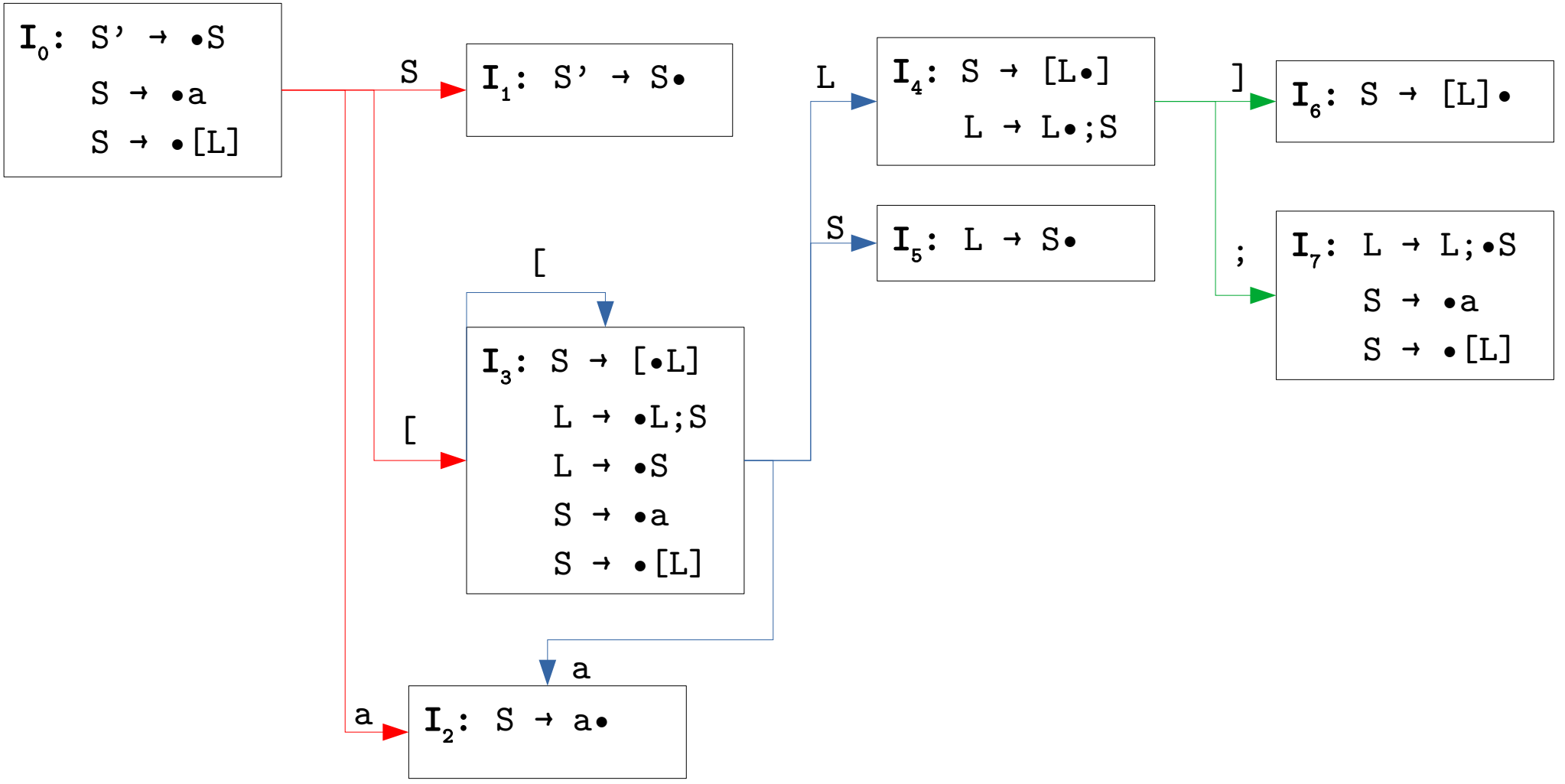
I_0: S' → •S

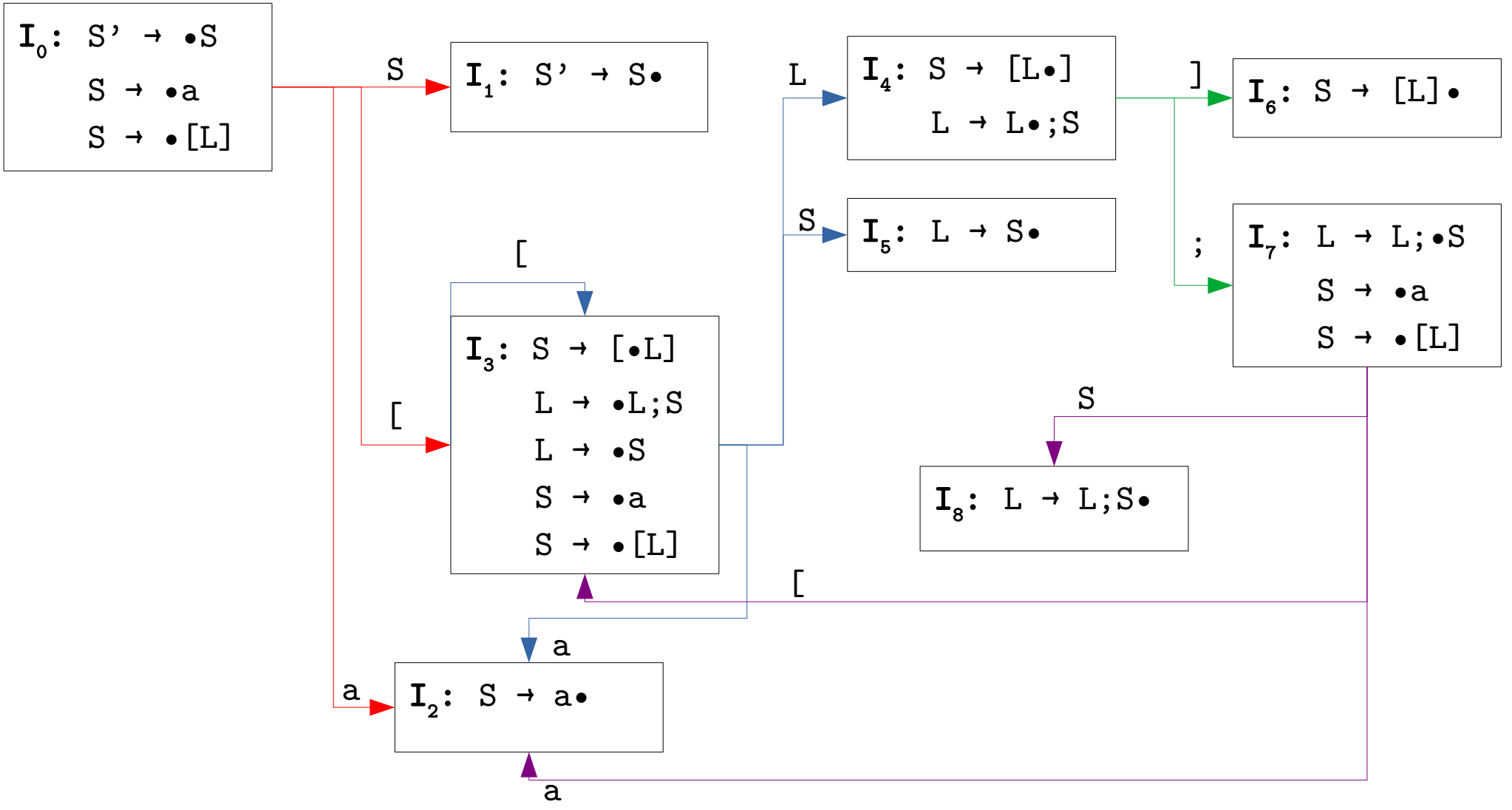
S → •a

S → •[L]


I0: S' → •S
S → •a
S → •[L]
S
I1: S' → S•
[
I3: S → [•L]
L → •L;S
L → •S
S → •a
S → •[L]
a
I2: S → a•

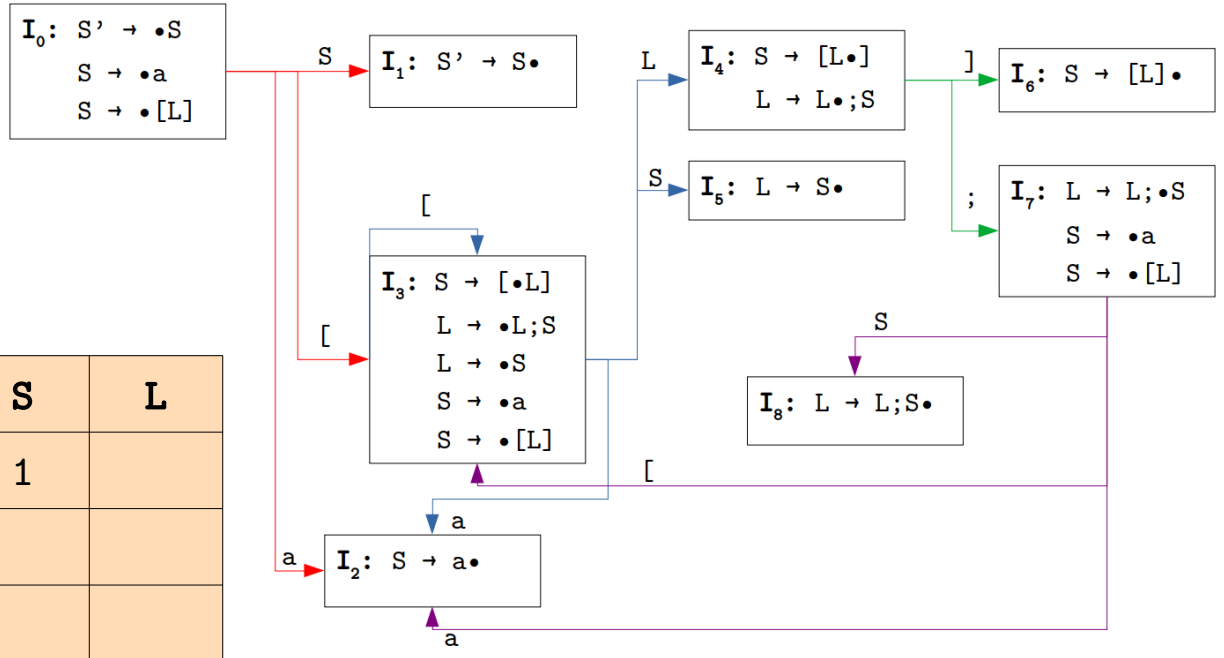

I0: S' → •S
S → •a
S → •[L]
S
I1: S' → S•
L
I4: S → [L•]
L → L•;S
S
I5: L → S•
[
I3: S → [•L]
L → •L;S
L → •S
S → •a
S → •[L]
[
a
I2: S → a•
a


I0: S' → •S
S → •a
S → •[L]
S
I1: S' → S•
[
I3: S → [•L]
L → •L;S
L → •S
S → •a
S → •[L]
a
I2: S → a•
L
I4: S → [L•]
L → L•;S
S
I5: L → S•
]
I6: S → [L]•
;
I7: L → L;•S
S → •a
S → •[L]


I0: S' → •S
S → •a
S → •[L]
S
I1: S' → S•
L
I4: S → [L•]
L → L•;S
]
I6: S → [L]•
S
I5: L → S•
;
I7: L → L;•S
S → •a
S → •[L]
[
I3: S → [•L]
L → •L;S
L → •S
S → •a
S → •[L]
[
S
I8: L → L;S•
[
a
a
I2: S → a•
a

Passo 3: Construção da tabela

	a	[	]	;	$	S	L
0	E2	E3				1	
1					AC		
2			R1	R1	R1		
3	E2	E3				5	4
4			E6	E7			
5			R4	R4			
6			R2	R2	R2		
7	E2	E3				8	
8			R3	R3			



0) S'→ S
1) S → a
2) S → [L]
3) L → L;S
4) L → S

Passo 4: Expansão da tabela para tratamento de erros

	a	[	]	;	$	S	L
0	E2	E3	erro	erro	erro	1	
1	erro	erro	erro	erro	AC		
2	R1	R1	R1	R1	R1		
3	E2	E3	erro	erro	erro	5	4
4	erro	erro	E6	E7	erro		
5	R4	R4	R4	R4	R4		
6	R2	R2	R2	R2	R2		
7	E2	E3	erro	erro	erro	8	
8	R3	R3	R3	R3	R3		

Passo 5: Reconhecimento da sentença (Exemplo 1)

Gramática:

0) S'→ S
1) S → a
2) S → [L]
3) L → L;S
4) L → S

	a	[	]	;	$	S	L
0	E2	E3	erro	erro	erro	1	
1	erro	erro	erro	erro	AC		
2	R1	R1	R1	R1	R1		
3	E2	E3	erro	erro	erro	5	4
4	erro	erro	E6	E7	erro		
5	R4	R4	R4	R4	R4		
6	R2	R2	R2	R2	R2		
7	E2	E3	erro	erro	erro	8	
8	R3	R3	R3	R3	R3		

Pilha	Entrada	Ação
$ 0	[[a;a]]$	E3
$ 0 [3	[a;a]]$	E3
$ 0 [3 [3	a;a]]$	E2
$ 0 [3 [3 a 2	;a]]$	R1
$ 0 [3 [3 S 5	;a]]$	R4
$ 0 [3 [3 L 4	;a]]$	E7
$ 0 [3 [3 L 4 ; 7	a]]$	E2
$ 0 [3 [3 L 4 ; 7 a 2	]]$	R1
$ 0 [3 [3 L 4 ; 7 S 8	]]$	R3
$ 0 [3 [3 L 4	]]$	**E6**
$ 0 [3 [3 L 4] 6	]$	R2
$ 0 [3 S 5	]$	**R4**
$ 0 [3 L 4	]$	E6
$ 0 [3 L 4] 6	$	R2
$ 0 S 1	$	**ACEITA!**

Passo 5: Reconhecimento da sentença (Exemplo 2)

Gramática:

0) S'→ S
1) S → a
2) S → [L]
3) L → L;S
4) L → S

	a	[	]	;	$	S	L
0	E2	E3	erro	erro	erro	1	
1	erro	erro	erro	erro	AC		
2	R1	R1	R1	R1	R1		
3	E2	E3	erro	erro	erro	5	4
4	erro	erro	E6	E7	erro		
5	R4	R4	R4	R4	R4		
6	R2	R2	R2	R2	R2		
7	E2	E3	erro	erro	erro	8	
8	R3	R3	R3	R3	R3		

Pilha	Entrada	Ação
$ 0	[a;]$	E3
$ 0 [3	a;]$	E2
$ 0 [3 a 2	;]$	R1
$ 0 [3 S 5	;]$	R4
$ 0 [3 L 4	;]$	E7
$ 0 [3 L 4 ; 7	]$	ERRO1
$ 0 [3 L 4 ; 7 a 2	]$	R1
$ 0 [3 L 4 ; 7 S 8	]$	R3
$ 0 [3 L 4	]$	E6
$ 0 [3 L 4] 6	$	**R2**
$ 0 S 1	$	ACEITA(!)

ERRO1: empilha um 'a' o '2' (estado 7 com a).
Mensagem: " *] não esperado*". *Espera-se 'a' ou '['.*

Análise Sintática Top-Down:

Exercícios

- **Exercício em duplas:**

  1)Construir a tabela sintática com tratamento de erros para a gramática abaixo:

  S → if E then C | C

  E → a

  C → b

  2) reconhecer as cadeias

  "*if a then b*"

  "*if a b*"