UDESC

Compiladores

Prof. Ricardo

ricardo.martins@udesc.br

UDESC

Métodos *bottom-up*

Podem ser vistos como a tentativa de se reduzir a cadeia de entrada ao símbolo inicial da gramática.

Exemplos:

- Precedência de Operadores;
- SLR(1), LR(1), LALR(1).

UDESC

Métodos LR(*k*)

Os métodos de análise sintática LR executam uma derivação mais a direita ao contrário. O L significa que a varredura da entrada e feita da esquerda para a direita (*left to right*), o R que a derivação correspondente é a derivação mais a direita (*rightmost derivation*) e o *k* indica o número de símbolos de entrada que tem que ser examinados para se tomar uma decisão na análise sintática.

A diferença entre os métodos SLR e LALR é apenas a técnica usada para a construção das tabelas sintáticas.

Métodos LR

Exemplo: SLR(1)

Estado	AÇÃO						DESVIO		
	c	+	*	(	)	#	E	T	F
0	E5			E4			1	2	3
1		E6				ACEITA			
2		R2	E7		R2	R2			
3		R4	R4		R4	R4			
4	E5			E4			8	2	3
5		R6	R6		R6	R6			
6	E5			E4				9	3
7	E5			E4					10
8		E6			E11				
9		R1	E7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

(1) $E \rightarrow E + T$

(2) $E \rightarrow T$

(3) $T \rightarrow T * F$

(4) $T \rightarrow F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow c$

Algoritmo LR(1)

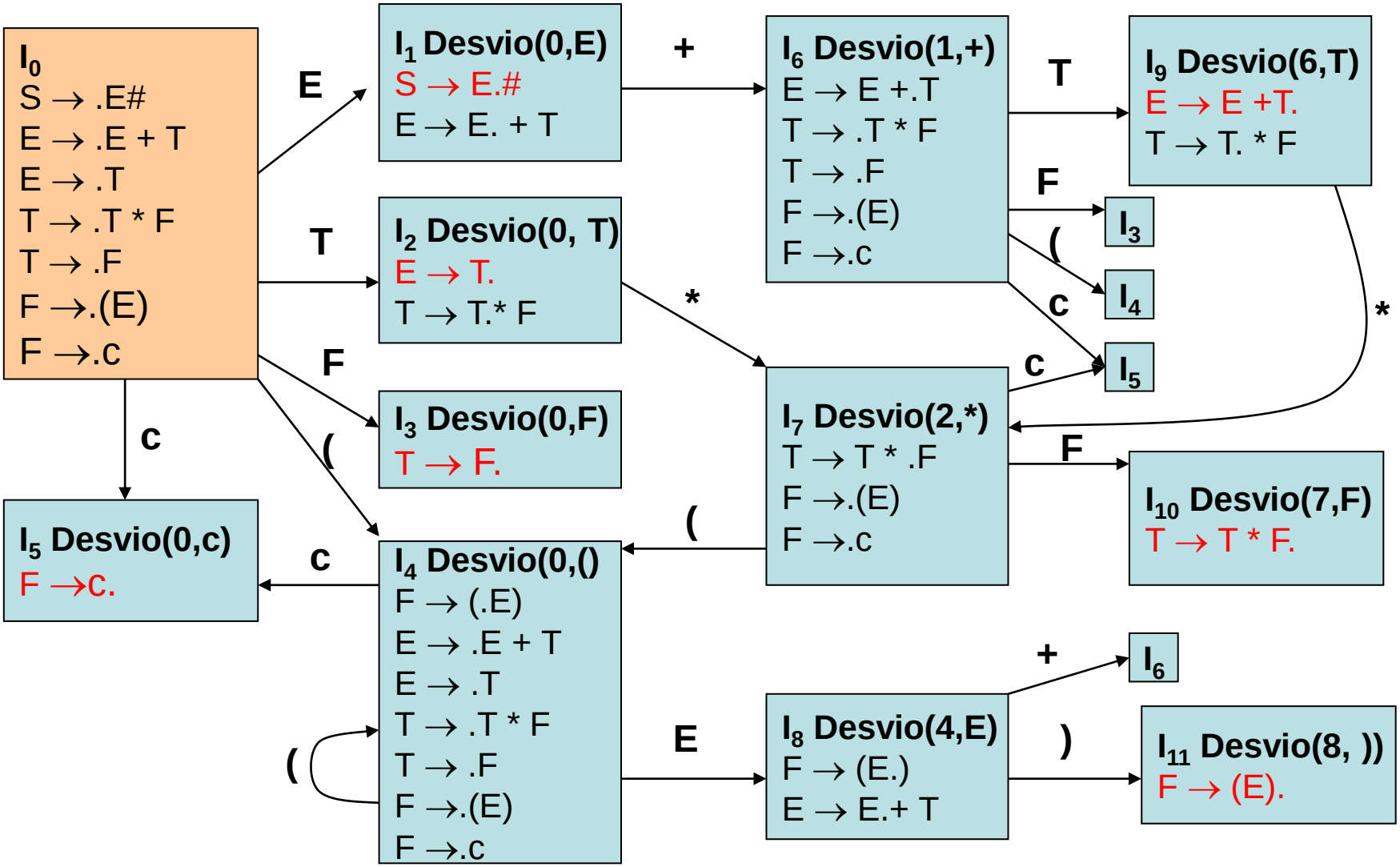
Considerando *w* a cadeia de entrada.
Empilhar 0. /* Estado inicial */
Faça *p* apontar para o primeiro símbolo de *w#*
Repetir para sempre
    Seja *s* o estado no topo da Pilha e *a* o símbolo apontado por p
    Se **AÇÃO**[*s*, *a*] = empilhar *s'* então
        Empilhar *a*; Empilhar *s'*;
        Avançar *p*;
    Senão
        Se **AÇÃO**[*s*, *a*] = reduzir $A \rightarrow \beta$ então
            Desempilhar 2 * $|\beta|$ símbolos;
            Seja *s'* o estado no topo da pilha
            Empilhar A; Empilhar DESVIO[*s'*, *A*];
        Senão
            Se **AÇÃO**[*s*, *a*] = aceitar então Retornar;
            Senão erro
fim

UDESC

Pilha : 0

Entrada: c + c #

Exemplo: SLR(1)

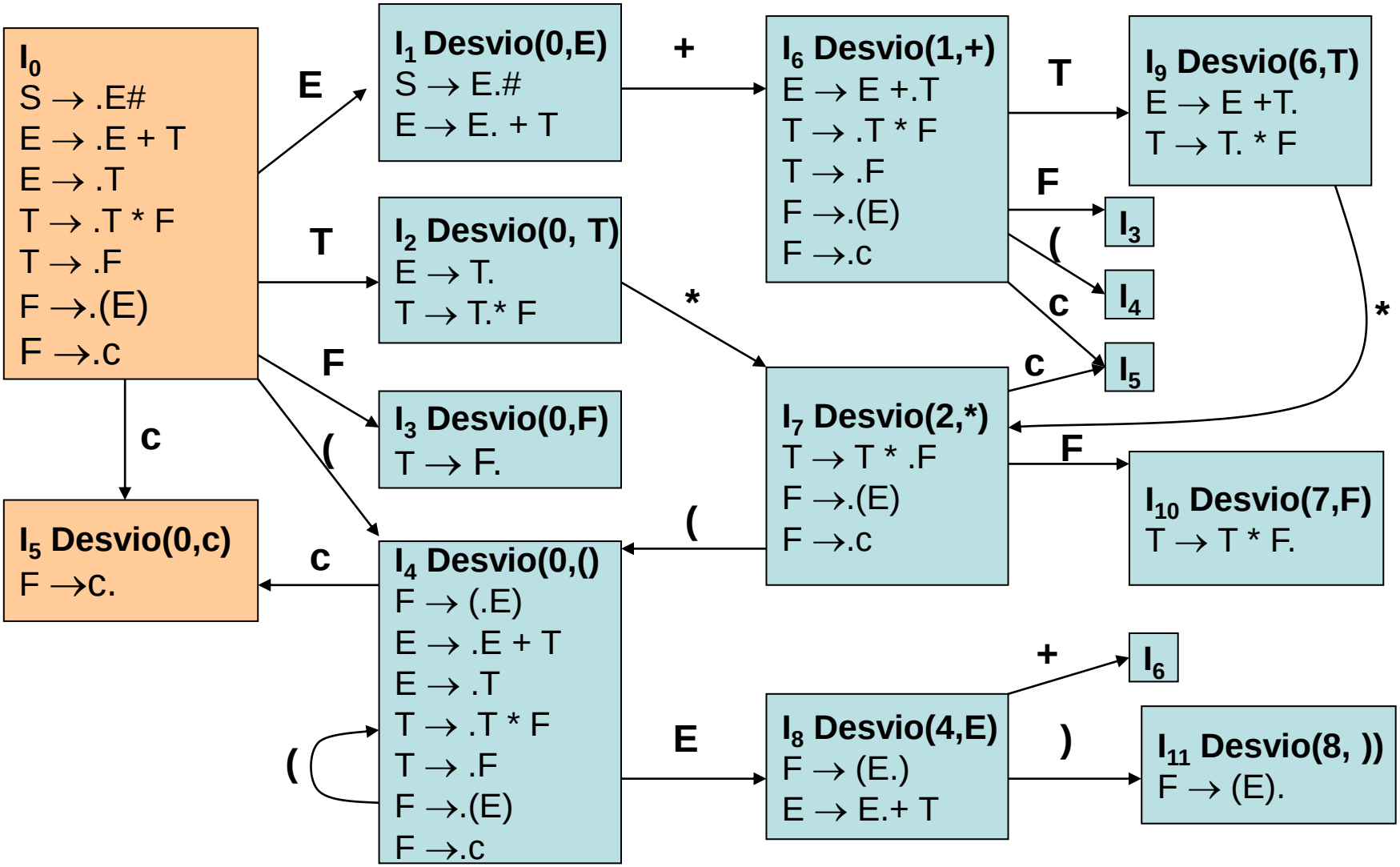


UDESC

Pilha : 0 c 5

Entrada: c + c #

Exemplo: SLR(1)

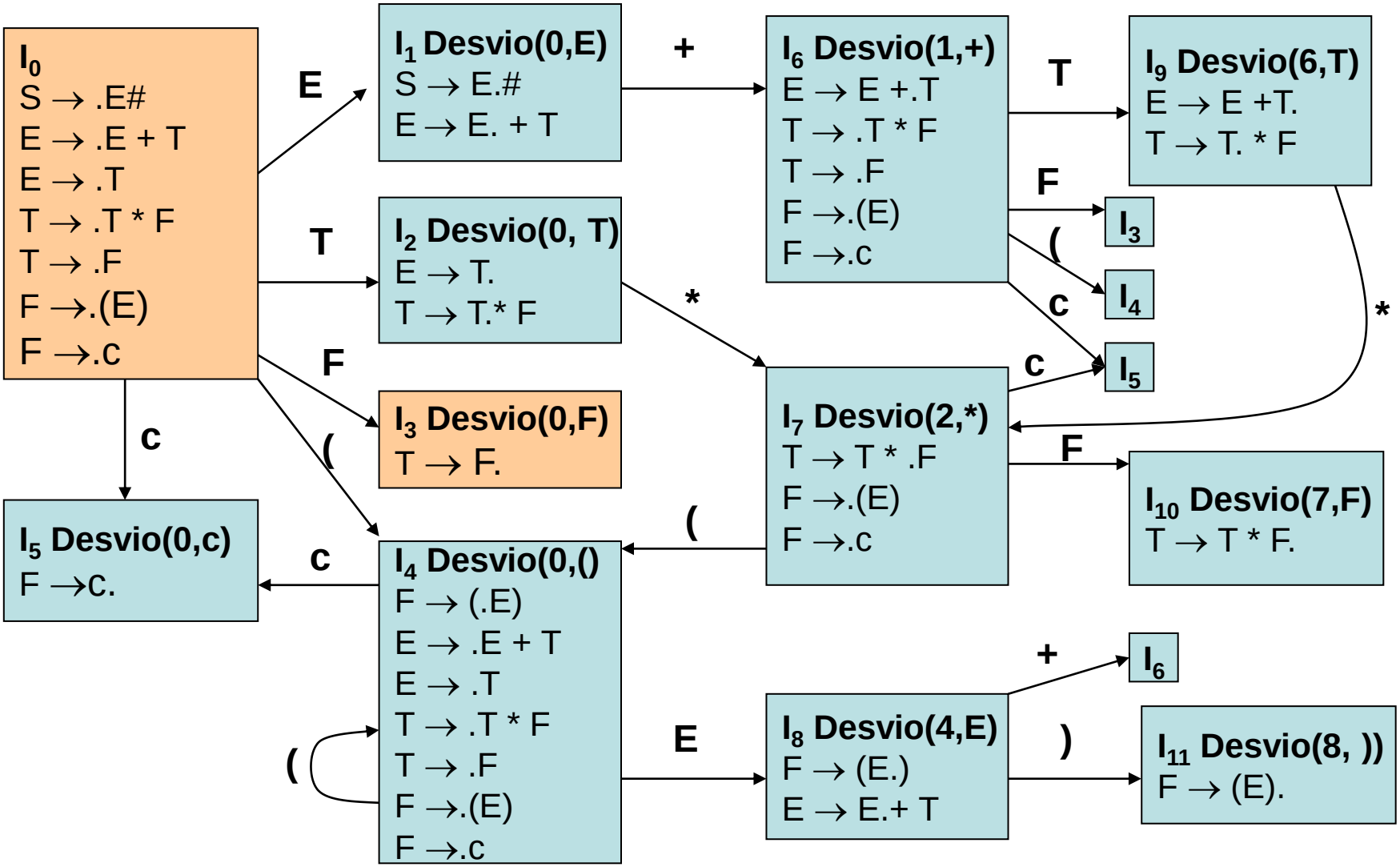


UDESC

Pilha : 0 F 3

Entrada: c + c #

Exemplo: SLR(1)

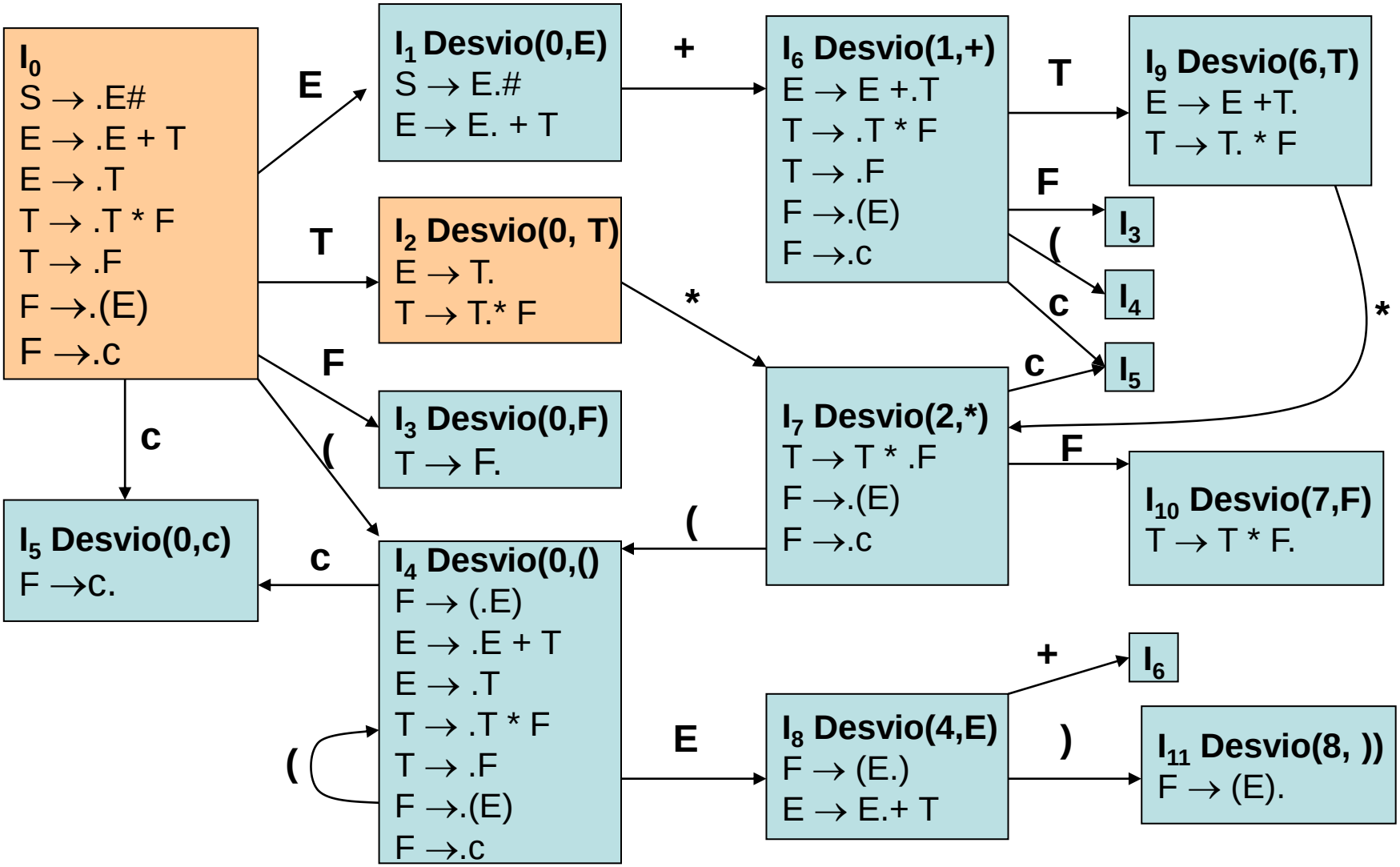


UDESC

Pilha : 0 T 2

Entrada: c + c #

Exemplo: SLR(1)

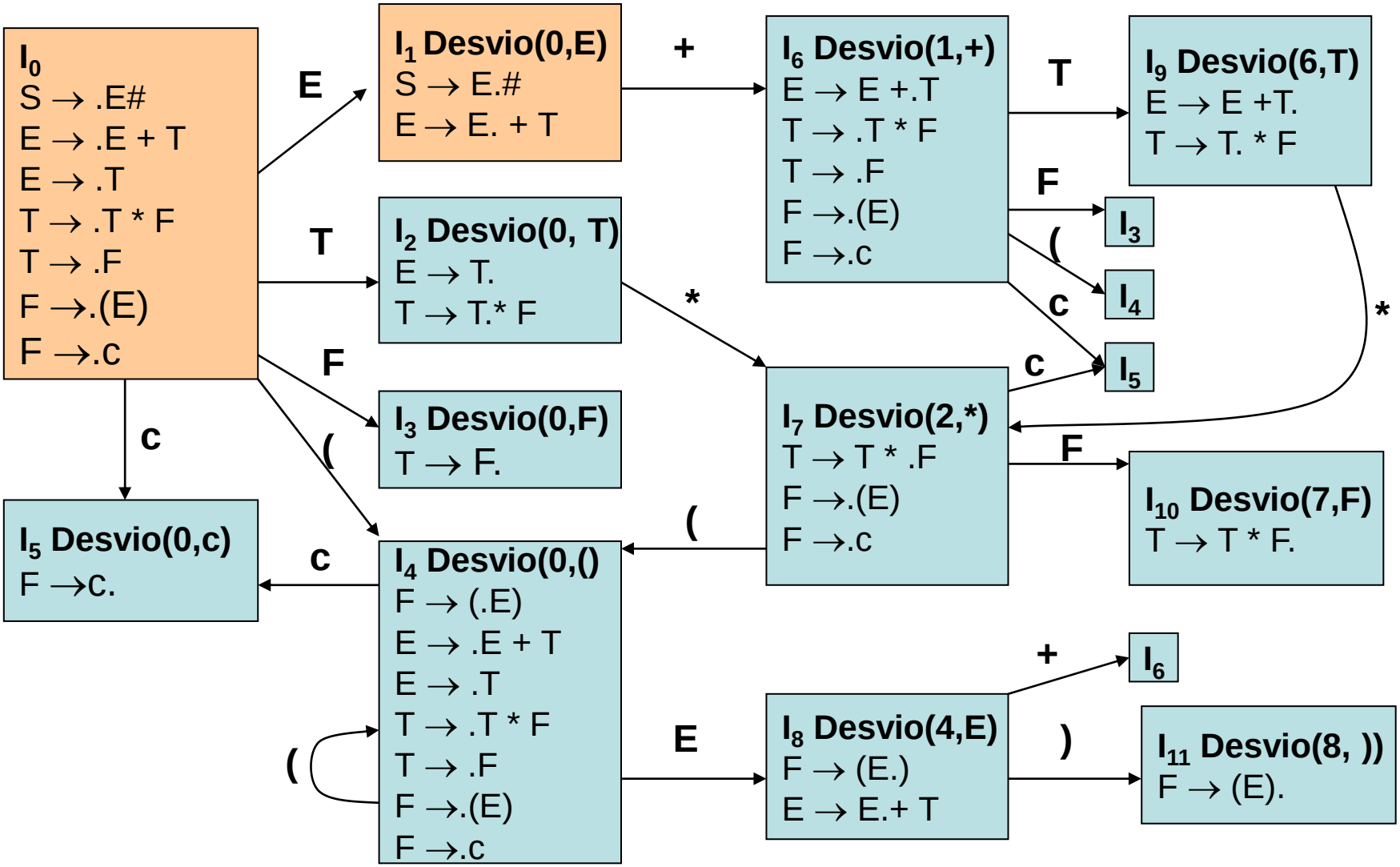


UDESC

Pilha : 0 E 1

Entrada: c + c #

Exemplo: SLR(1)

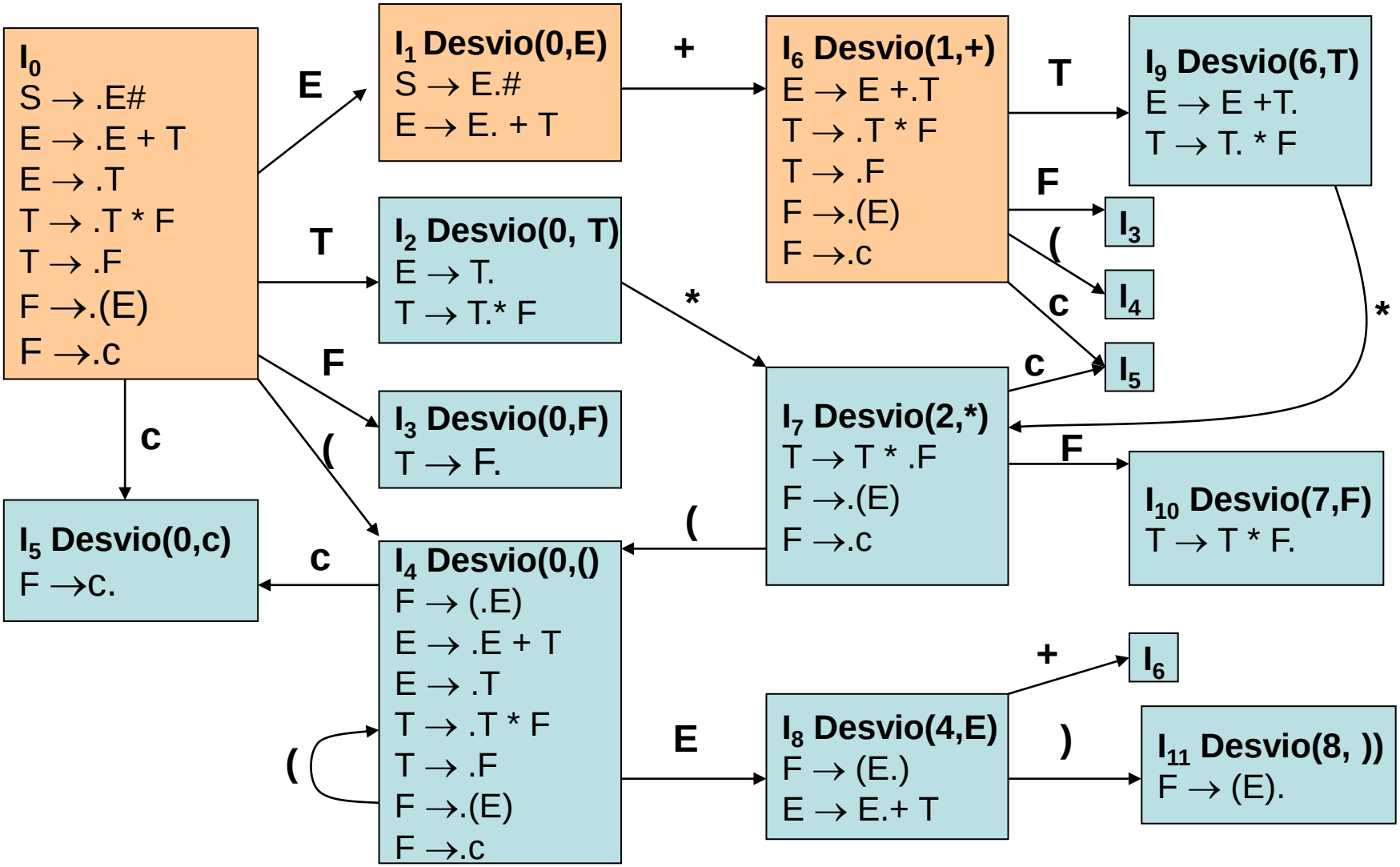


UDESC

Pilha : 0 E 1 + 6

Entrada: c + c #

Exemplo: SLR(1)

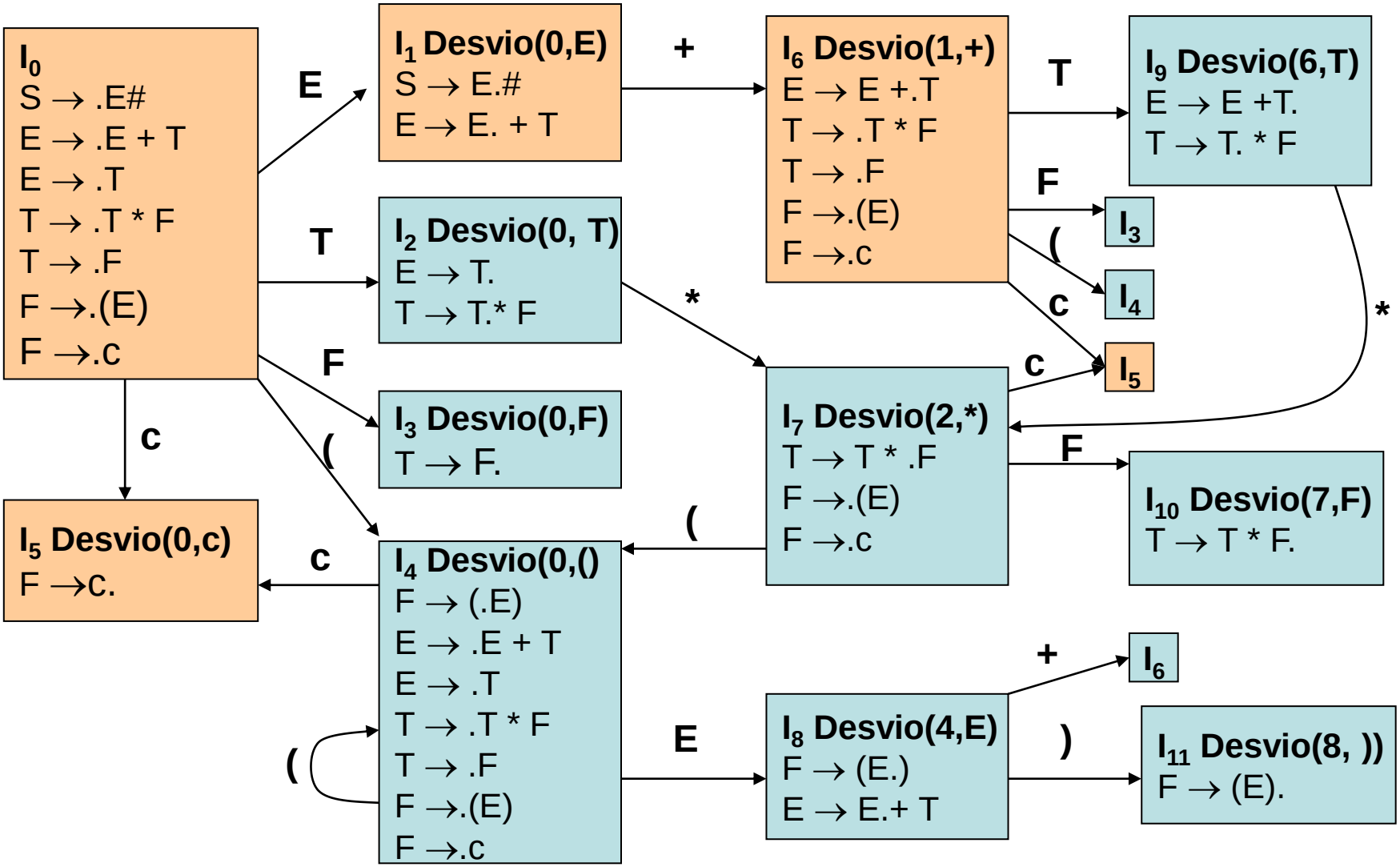


UDESC

Pilha : 0 E 1 + 6 c 5

Entrada: c + c #

Exemplo: SLR(1)

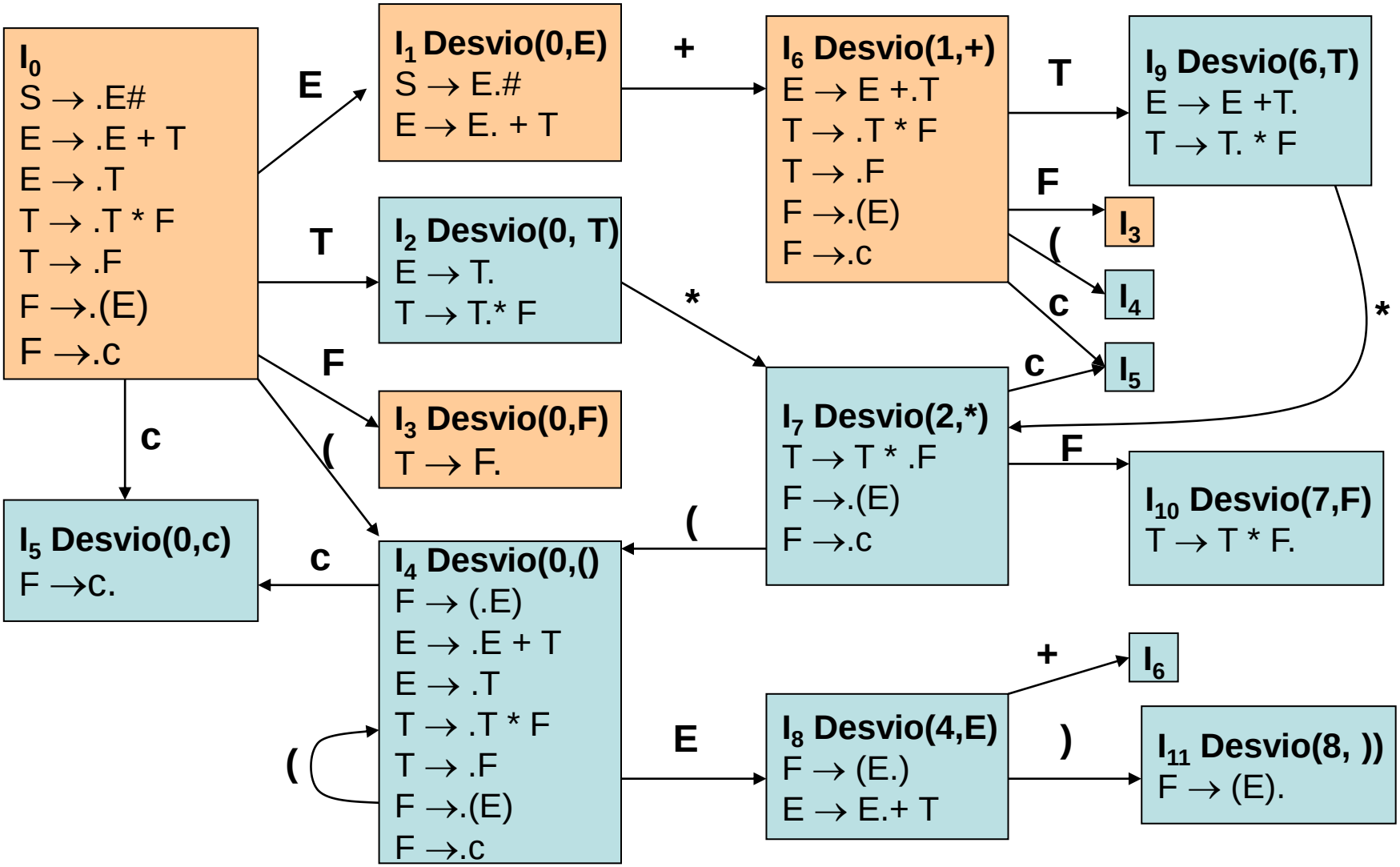


UDESC

Pilha : 0 E 1 + 6 F 3

Entrada: c + c #

Exemplo: SLR(1)

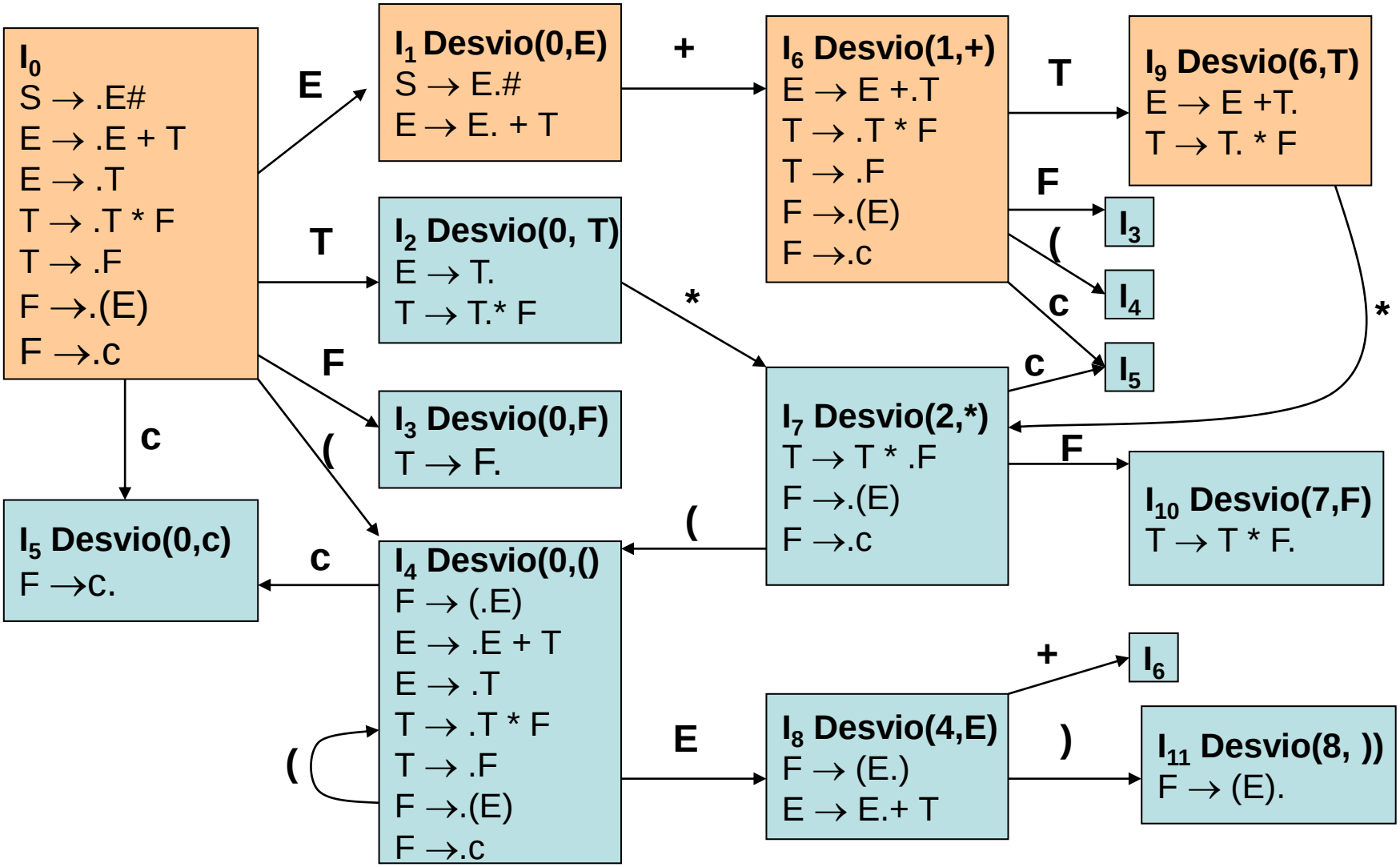


UDESC

Pilha : 0 E 1 + 6 T 9

Entrada: c + c #

Exemplo: SLR(1)

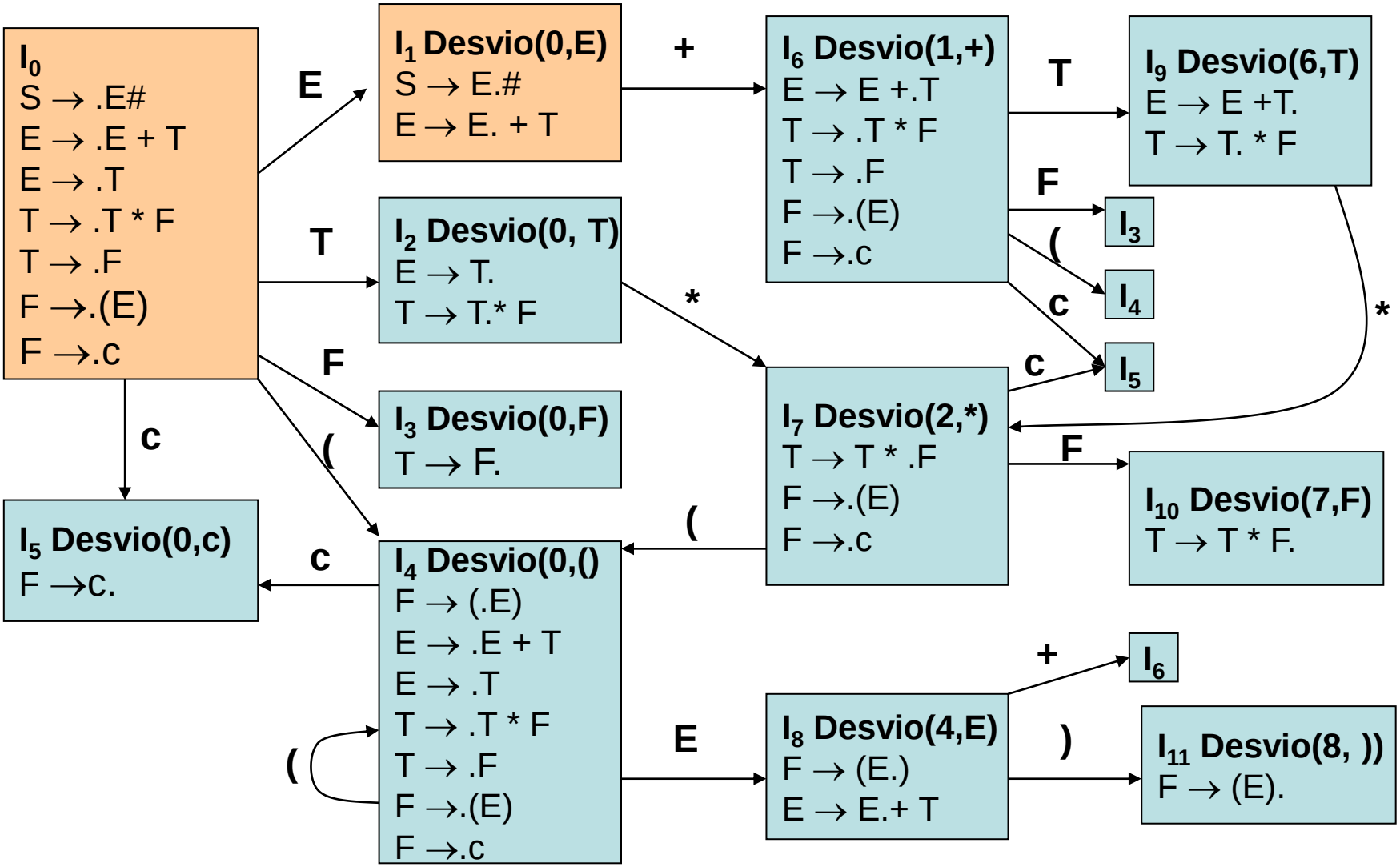


UDESC

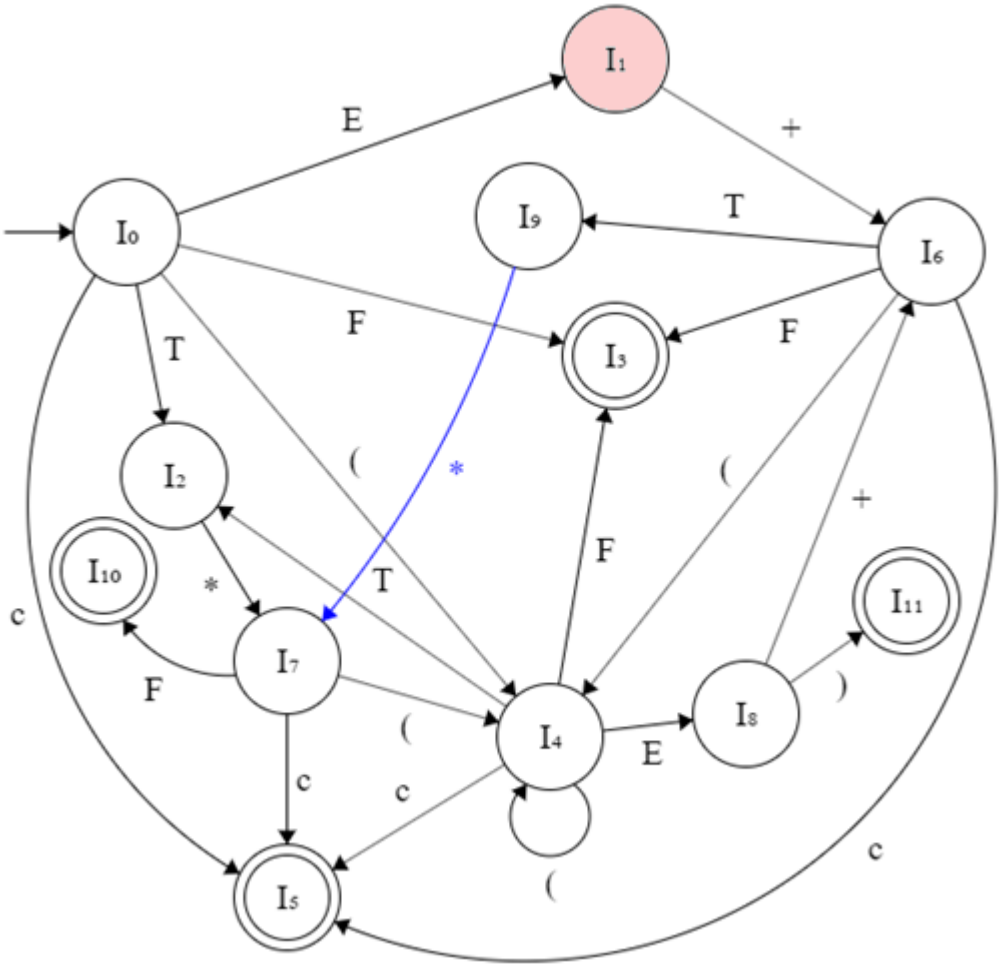
Pilha : 0 E 1

Entrada: c + c #

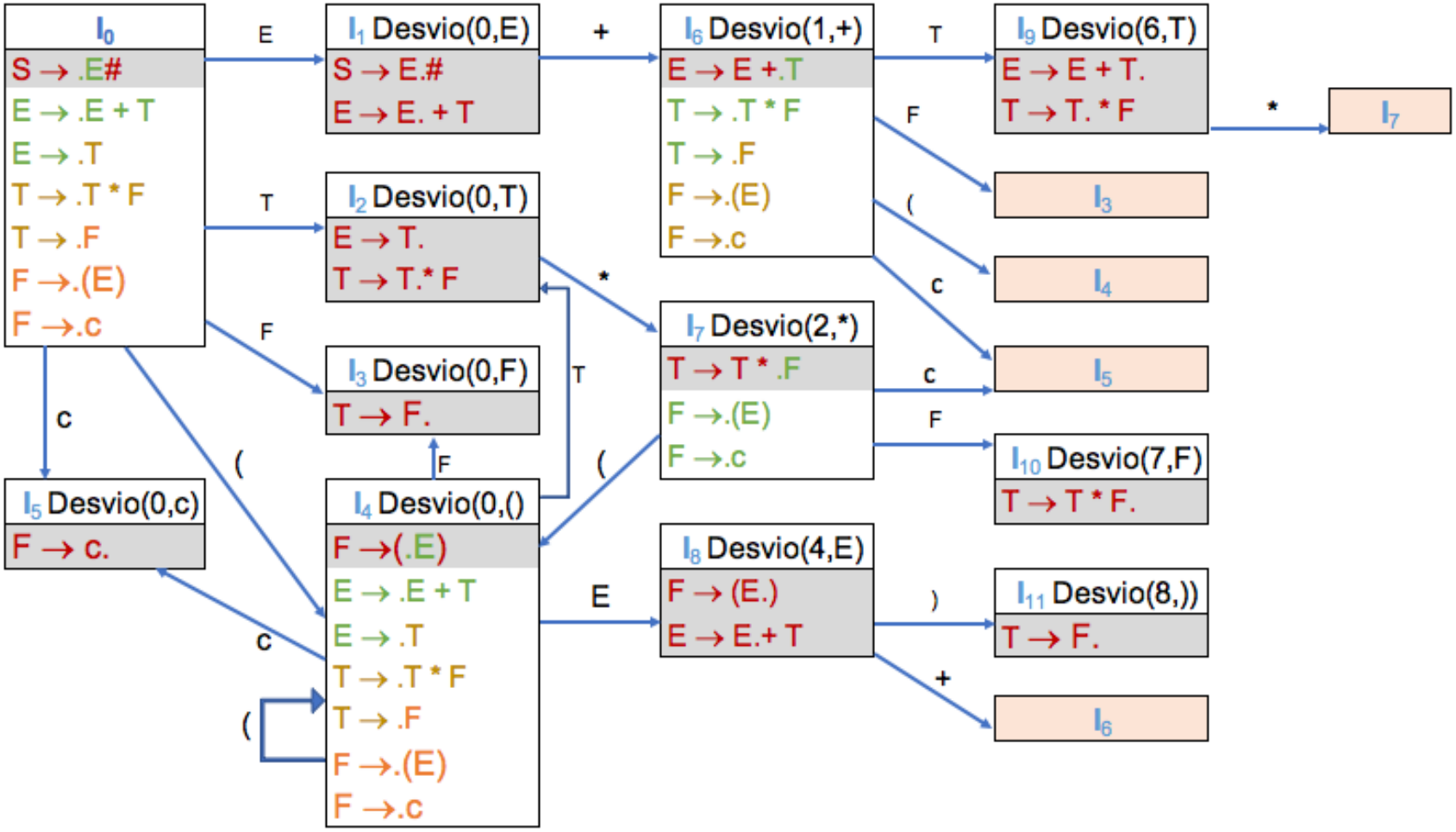
Exemplo: SLR(1)



Modelo do Autômato - SLR(1)



Autômato completo - SLR(1)



UDESC

Tabelas SLR(1)

Um item LR(0), para uma gramática *G*, é uma produção de *G* com um ponto em alguma das posições do seu lado direito.

Exemplo: A produção E $\rightarrow$ E + T produz 4 itens:

[E $\rightarrow$ **.**E + T]

[E $\rightarrow$ E **.**+ T]

[E $\rightarrow$ E +**.**T]

[E $\rightarrow$ E + T**.**]

Intuitivamente o **.** indica até que parte da produção foi analisada em um determinado estado do analisador sintático.

Construção da Tabela Sintática SLR(1)

A construção da tabela sintática é auxiliada por duas operações:

Fechamento:

Sendo I um conjunto de itens da gramática, o fechamento de I é definido por duas regras:

1. Inicialmente cada item de *I* é adicionado em FECHAMENTO(*I*).
2. Se [$A \rightarrow \alpha.B\beta$] estiver em FECHAMENTO(*I*) e $B \rightarrow \gamma$ for uma produção, adicionar o item [$B \rightarrow .\gamma$] a FECHAMENTO(*I*). Essa regra é aplicada até que nenhum novo item possa ser adicionado.

Desvio:

A operação DESVIO(*I*, *X*) é definida como o fechamento do conjunto de todos os itens [$A \rightarrow \alpha X.\beta$] tais que [$A \rightarrow \alpha.X\beta$] esteja em *I*.

Construção da Tabela Sintática SLR(1)

Construção de um conjunto de itens LR(0):

C ← FECHAMENTO(*S'* → .*S*#) /* Onde *S* é o símbolo inicial da linguagem */
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*) em *C*
Até que não haja mais conjunto de itens a serem incluídos em *C*

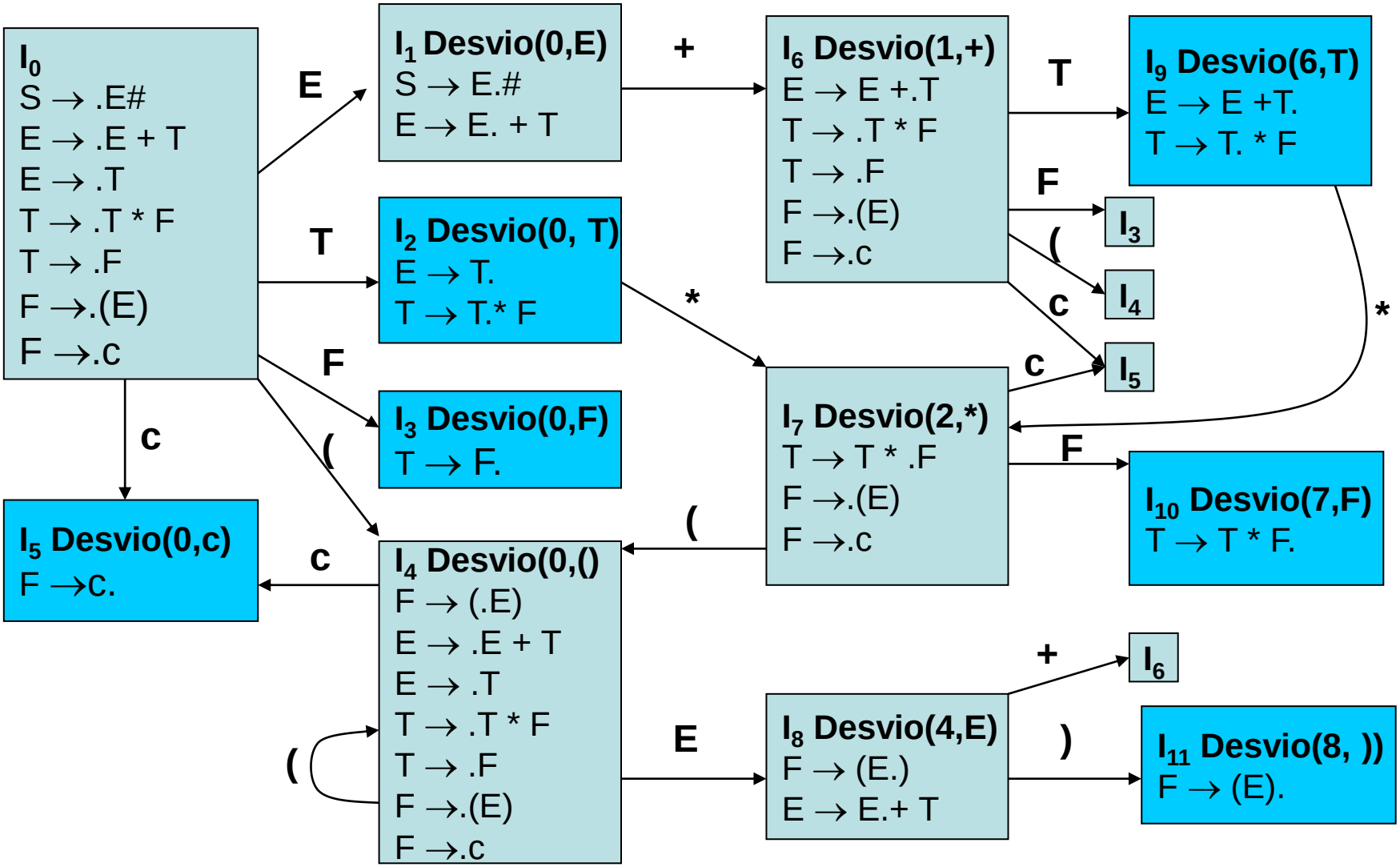
Construção da Tabela Sintática SLR(1)

Construção da tabela sintática SLR(1):

1. Construir o conjunto de itens $C = \{I_0, I_1, ..., I_n\}$
2. Cada estado i é construído a partir de I_i. As ações sintáticas são determinadas como:

- Se $[A \rightarrow \alpha.a\beta]$ estiver em I_i e DESVIO(I_i, a) = I_j então ação[i, a] = Empilhar j (se a for um terminal) ou desvio[i, a] = Empilhar j (se a for um não terminal)
- Se $[A \rightarrow \alpha.]$ estiver em I_i então ação(i, a) = reduzir através de $A \rightarrow \alpha$, para todo a em SEGUINTES(A).
- Se $[S' \rightarrow S.\#]$ estiver em I_i então ação(i, #) = aceitar

Construção da Tabela Sintática SLR(1)



Construção da Tabela Sintática SLR(1)

I_5 Desvio(0,c) F →c.	SEGUINTES(F) = { +, *,), # }
I_2 Desvio(0, T) E → T. T → T.* F	SEGUINTES(E) = { +,), # }
I_3 Desvio(0,F) T → F.	SEGUINTES(T) = { +, *,), # }
I_9 Desvio(6,T) E → E +T. T → T. * F	SEGUINTES(E) = { +,), # }
I_{10} Desvio(7,F) T → T * F.	SEGUINTES(F) = { +, *,), # }
I_{11} Desvio(8,)) F → (E).	SEGUINTES(F) = { +, *,), # }

Se [$A \rightarrow \alpha.$] estiver em I_i então ação(i, a) = redução através de $A \rightarrow \alpha$, para todo a em SEGUINTES(A).

UDESC

Construção da Tabela Sintática SLR(1)

No método SLR(1), como a decisão de reduzir aplicando uma produção $A \rightarrow \alpha$, é tomada usando o conjunto SEGUINTES(A) e não o contexto onde α ocorreu, algumas gramáticas LR(1) podem apresentar conflitos empilhar/reduzir se tentamos construir as tabelas usando esse método. Por o exemplo:

$S \rightarrow L = R$

$S \rightarrow R$

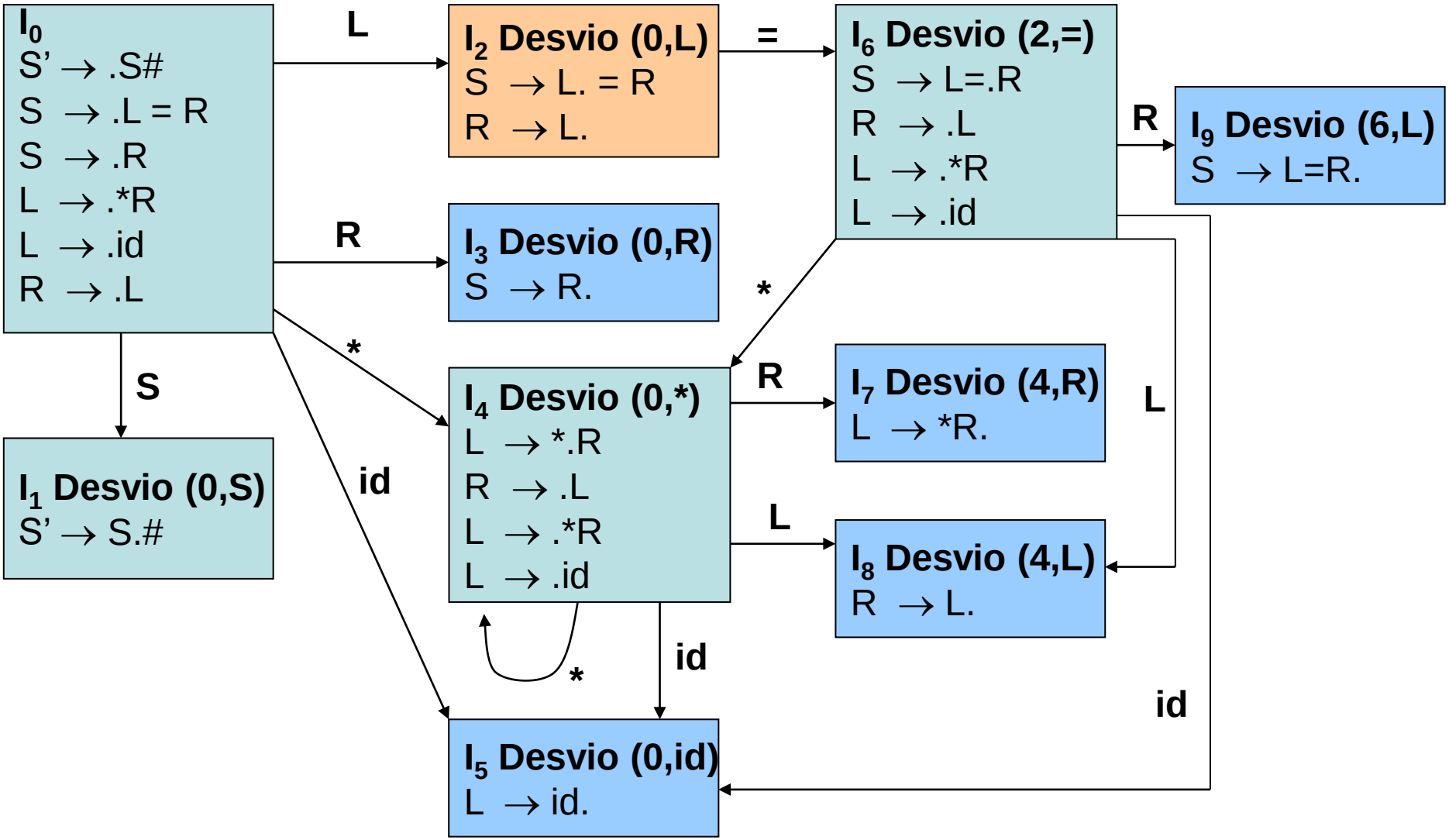
$L \rightarrow *R$

$L \rightarrow$ id

$R \rightarrow L$

*(Nesse exemplo os não terminais L e R representam **l-value** e **r-value** respectivamente)*

Construção da Tabela Sintática SLR(1)



Métodos LR

Exemplo: SLR(1)

Estado	AÇÕES				DESVIO		
	*	=	id	#	S	L	R
0	E4		E5		1	2	3
1				aceita			
2		E6 / R5		R5			
3				R2			
4	E4		E5			8	7
5		R4		R4			
6	E4		E5			8	9
7		R3		R3			
8		R5		R5			
9				R1			

(1) $S \rightarrow L = R$
(2) $S \rightarrow R$
(3) $L \rightarrow *R$
(4) $L \rightarrow$ id
(5) $R \rightarrow L$

Construção da Tabela Sintática SLR(1)

I_2 Desvio (0,L)
S $\rightarrow$ L. = R
R $\rightarrow$ L.

SEGUINTES(R) = {=, # }

No estado 2, a ação reduzir R $\rightarrow$ L deve ser executada para todos os seguintes de R, o que nesse caso ocasiona um conflito empilhar/reduzir. Entretanto não existe forma sentencial da gramática que inicie com **R =**. Para tratar essa gramática é necessário um método que carregue mais informação sobre o contexto para o estado.

Construção da Tabela Sintática LR(1)

Fechamento(*I*):

Repetir

Para cada item [$A \rightarrow \alpha.B\beta$, a] em I, cada produção $B \rightarrow \gamma$ na Gramática

e cada termina b em PRIMEIROS(βa) tal que [$B \rightarrow .\gamma$, b] não esta em I

Faça Incluir [$B \rightarrow .\gamma$, b] em I

até que não seja mais possível adicionar itens a I.

Desvio(*I*, *X*): é fechamento do conjunto de itens [$A \rightarrow \alpha X.\beta$, a] tais que

[$A \rightarrow \alpha.X\beta$, a] esta em I.

Itens(*G'*):

C = {FECHAMENTO({[$S' \rightarrow .S\#$, ε]})} (*Onde S é o símbolo inicial da gramática*)

Repetir

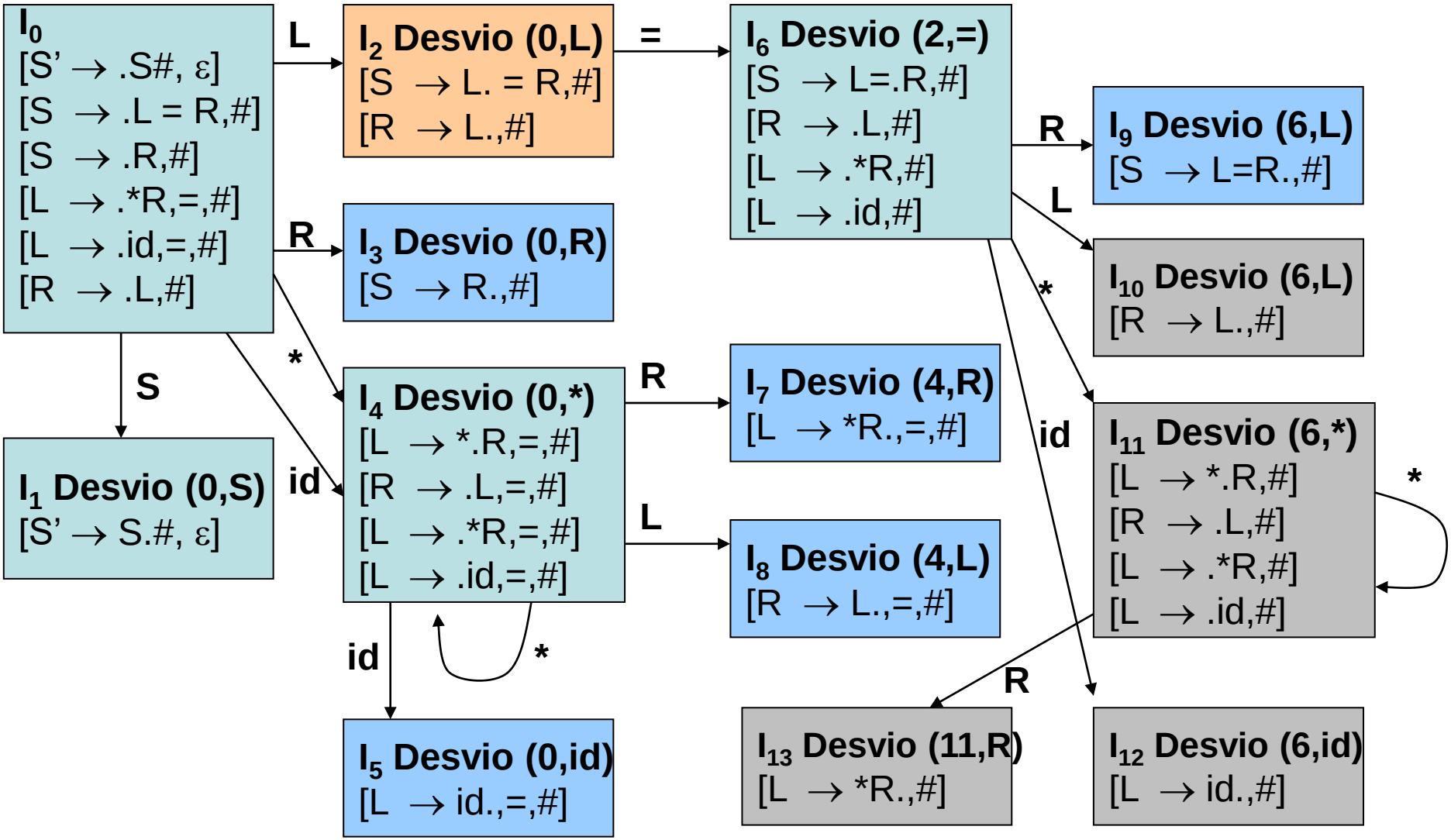
Para cada conjunto de itens I em C e cada símbolo gramatical X tal que

Desvio(I, X) $\neq \emptyset$ e Desvio(I, X) $\notin C$

Faça Incluir Desvio(I, X) em C

até que nenhum novo item possa ser adicionado a C

Construção da Tabela Sintática LR(1)



UDESC

LALR – lookahead LR

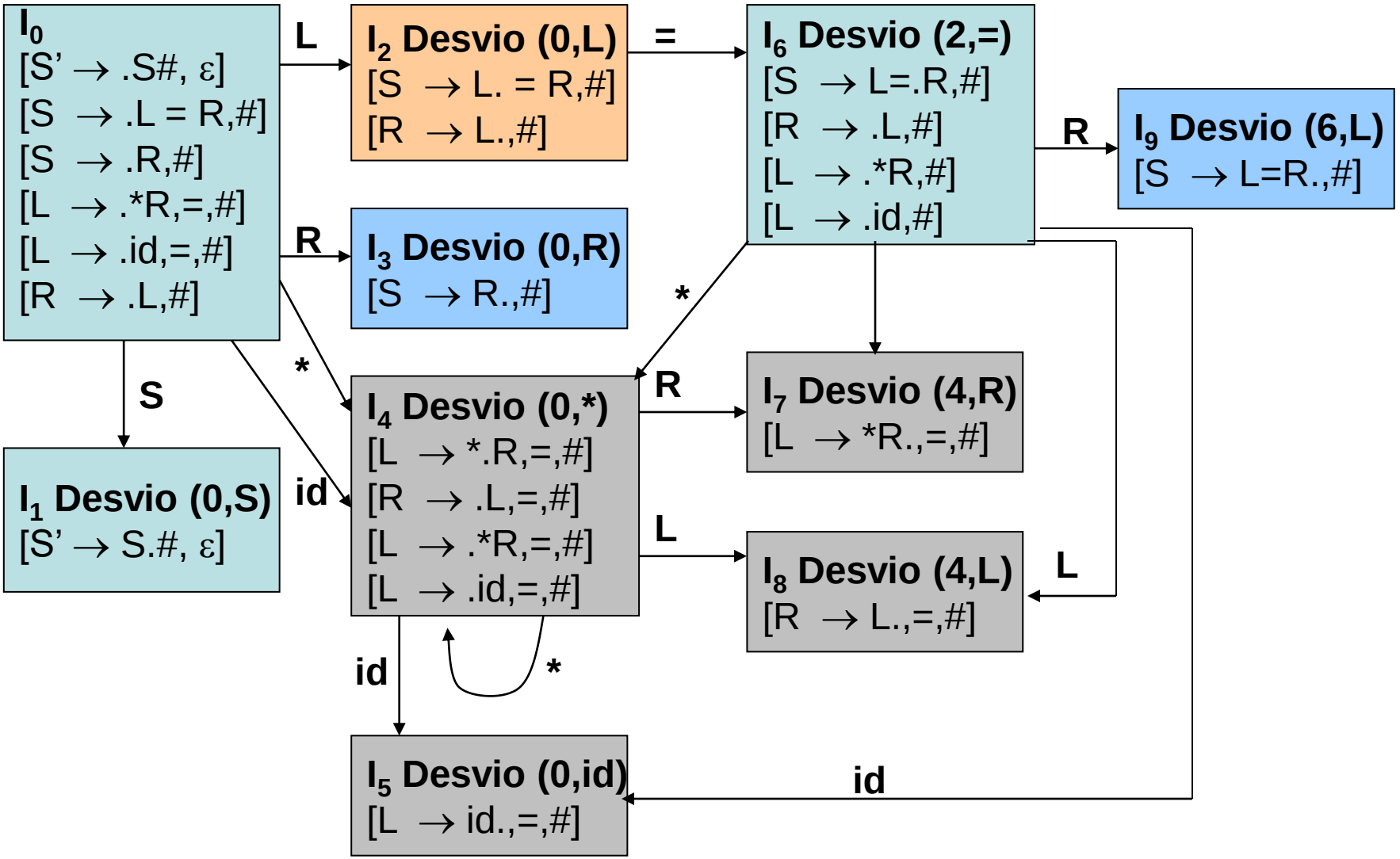
A idéia geral é construir o conjunto de itens LR(1) e, se nenhum conflito aparecer, combinar os itens com núcleo comum.

Algoritmos eficientes para a geração de tabelas LALR constroem o conjunto de itens LR(0) e numa segunda etapa determinam os *lookaheads* correspondentes de cada item.

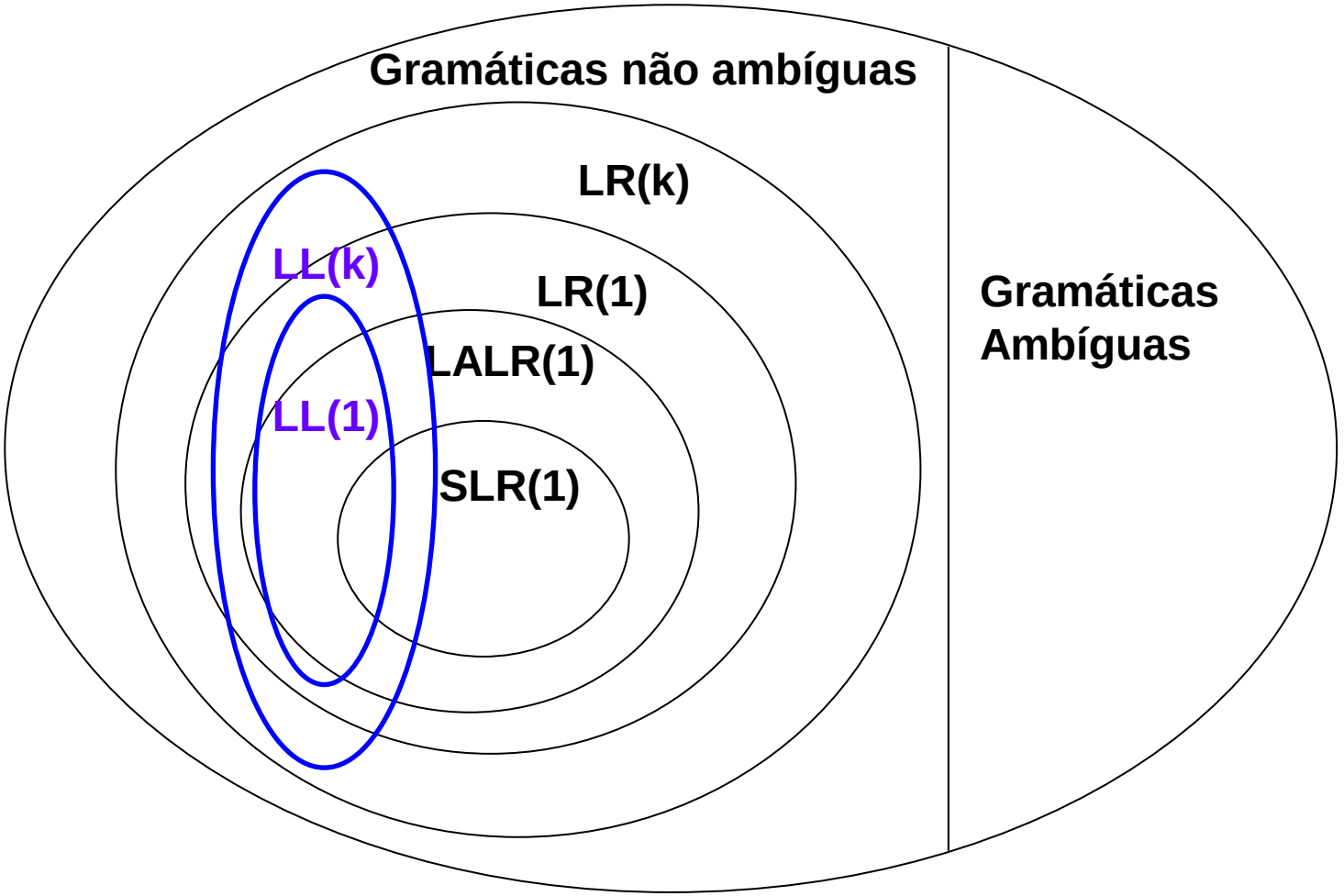
O método LALR:

- Trata a maioria dos casos presentes em linguagens de programação;
- Na grande maioria dos casos o número de estados é muito inferior ao número de estados gerados pelo método LR(1).
- Comparando com o método LR(1), em algumas situações, a detecção de erro é postergada, reduções desnecessárias são aplicadas antes que o erro seja detectado.

Construção da Tabela Sintática LALR(1)



Hierarquia de Gramáticas Livres de Contexto

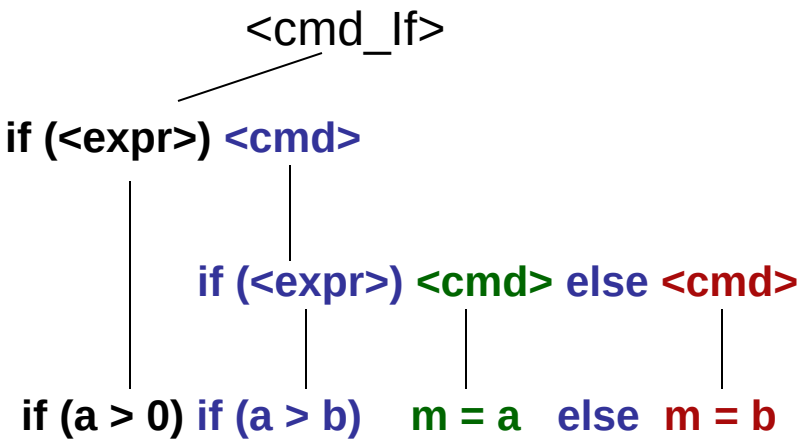


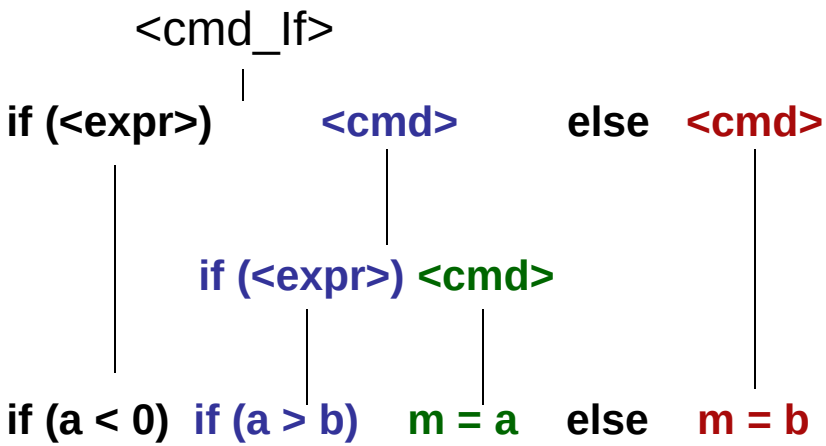
Uso de Gramáticas Ambíguas

<cmd_If> → if (<expr>) <cmd> else <cmd>

| if (<expr>) <cmd>

<cmd> → ... | <cmd> | ...





Essa gramática é ambígua, como consequência temos um conflito ***empilhar/reduzir*** *no estado em que ocorre a transição para o token* ***"else"****. Caso esse conflito seja resolvido escolhendo a opção* ***empilhar*** *as reduções executadas serão as correspondentes a primeira árvore.*