Compiladores

Aula 8

Celso Olivete Júnior

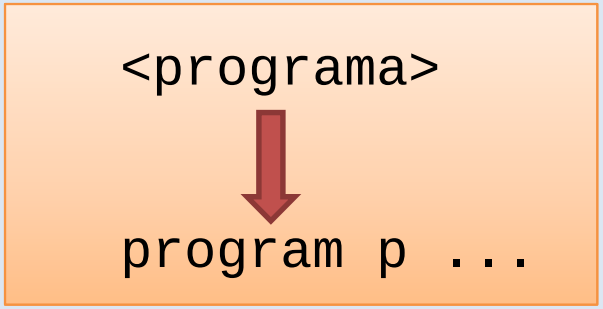
olivete@fct.unesp.br

Análise sintática
tipos de analisadores

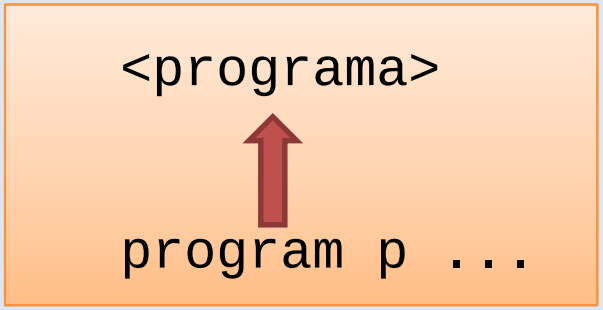
- Analisadores TOP-DOWN:
  - Árvore de derivação da raiz às folhas ➔ Análise Descendente
  - Tipo LL(1): Left to right / Leftmost derivation / 1 symbol each time - lookahead
  - Recursivo com Retrocesso (Backtracking)
  - Preditivo Tabular (não recursivo - pilha + tabela)
  - Tratamento de erros
- **Analisadores BOTTON-UP:**
  - **Árvore de derivação das folhas à raiz ➔ Análise Ascendente**
  - **Shift / Reduce - Análise Redutiva**
  - **LR(k) ➔**
    - ✓ **i) SLR (simple)**
    - ✓ **ii) LR canônicos**
    - ✓ **iii) LALR (lookahead LR)**

Análise sintática
tipos de analisadores

- Top-down ou descendente
  - Da raiz para as folhas



- Bottom-up ou ascendente
  - Das folhas para a raiz



Análise sintática ascendente

- Parte-se dos símbolos terminais em direção ao símbolo inicial da gramática

- Processo de derivação mais à direita

Análise sintática ascendente
exemplo

- Entrada **id * id**

E → E + T | T
T → T * F | F
F → (E) | id

id * id

Análise sintática ascendente
exemplo

- Entrada **id * id**

E → E + T | T
T → T * F | F
F → (E) | id

id * id	F * **id**
	|
	id

Análise sintática ascendente
exemplo

- Entrada **id * id**

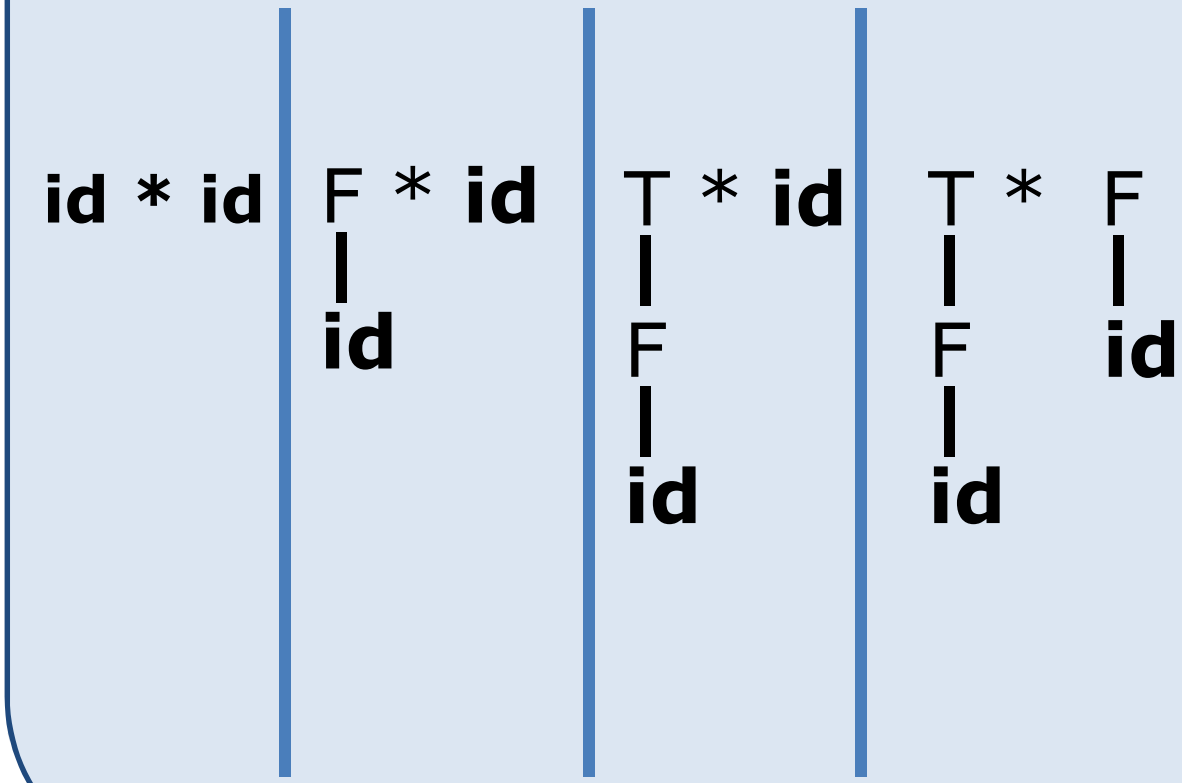
E → E + T | T
T → T * F | F
F → (E) | id

id * id	F * **id** | **id**	T * **id** | F | **id**

Análise sintática ascendente
exemplo

- Entrada **id * id**

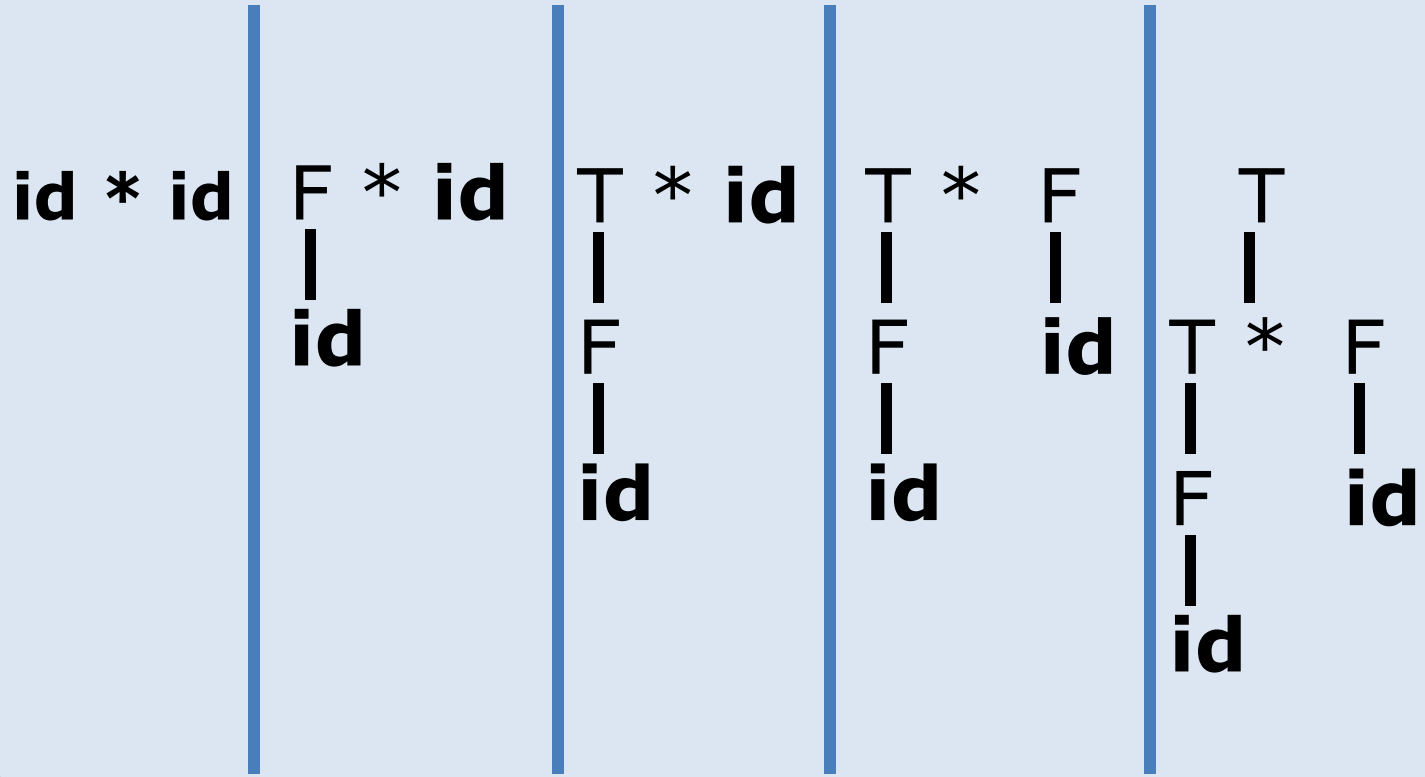
E → E + T | T
T → T * F | F
F → (E) | id



Análise sintática ascendente
exemplo

E → E + T | T
T → T * F | F
F → (E) | id

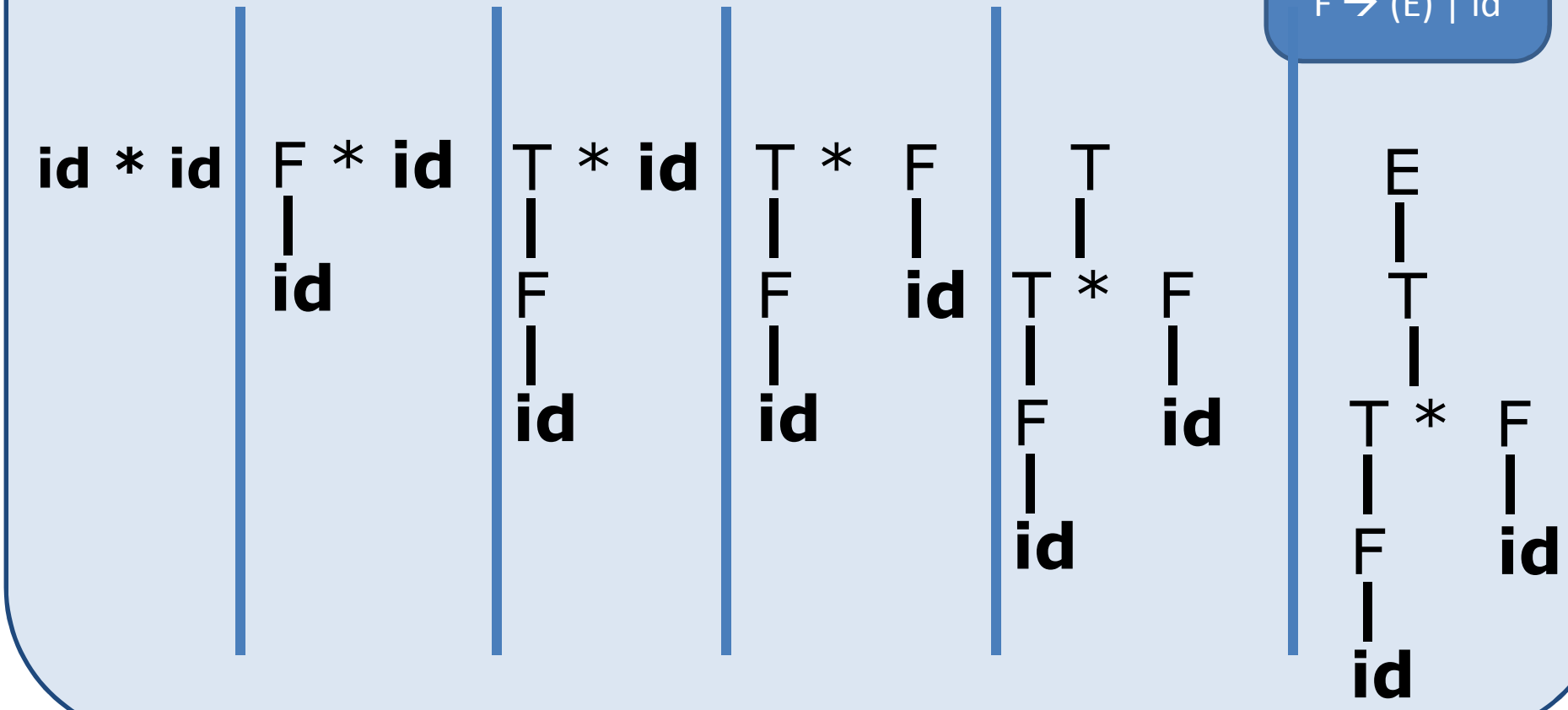
- Entrada **id * id**



Análise sintática ascendente
exemplo

- Entrada **id * id**

E → E + T | T
T → T * F | F
F → (E) | id



Análise sintática ascendente
Redução

- O processo de análise sintática ascendente pode ser encarado como um processo de "reduzir" uma cadeia *w* para o símbolo inicial da gramática

Análise sintática ascendente
Redução

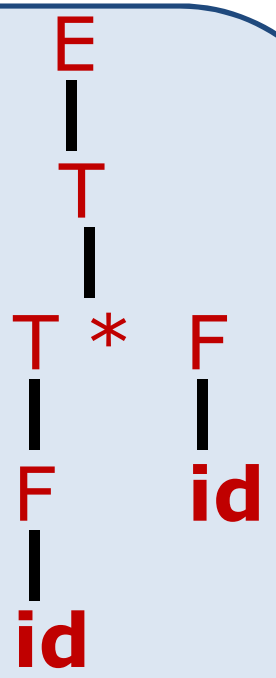
- **Redução:** operação de substituição do lado direito de uma produção pelo não-terminal correspondente do lado esquerdo
  - Para a regra $A \rightarrow \alpha$, α pode ser reduzido em A

Análise sintática ascendente
Redução

- Analisadores sintáticos ascendentes
  - Analisadores conhecidos como empilha-reduz (*shift-reduce*)
  - Etapas do reconhecimento: determinar quando *reduzir* e determinar a produção a ser utilizada para que a análise prossiga

Análise sintática ascendente
Redução

- Exemplo de sequências de reduções
  - id * id → F * id → T * id → T * F → T → E



E → E + T | T
T → T * F | F
F → (E) | id

Análise sintática ascendente

- Componentes do analisador ascendente
  - Pilha, onde os símbolos a serem reduzidos são empilhados
  - Tabela sintática que guia o processo de empilhamento e redução

- Processo de reconhecimento de uma sentença
  1. Empilhar símbolos da cadeia de entrada
  2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
  3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [1

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a 1

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a 2

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S 2

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ; 1

Análise sintática ascendente

- A partir da gramática

```
<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>
```

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a 1

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a [2]

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a
$[L;S	]$	Reduz L → L;S 2

Análise sintática ascendente

- A partir da gramática

```
<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>
```

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a
$[L;S	]$	Reduz L → L;S
$[L	]$	Empilha] 1

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a
$[L;S	]$	Reduz L → L;S
$[L	]$	Empilha]
$[L]	$	Reduz S → [L] 2

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a
$[L;S	]$	Reduz L → L;S
$[L	]$	Empilha]
$[L]	$	Reduz S → [L]
$S	$	**sucesso**

Análise sintática ascendente
funcionamento

- O analisador empilha símbolos até ter na pilha uma **sequência de símbolos** que corresponde à definição de um **não-terminal**
  - Sequência de símbolos: **lado direito da produção**
  - Não-terminal: **lado esquerdo da produção**

- *Handle*
  - Produção cujo lado direito está na pilha

- Operação de redução: substituição do lado direito do *handle* pelo seu lado esquerdo
  - O uso da sequência correta de *handles* no processo de análise leva ao símbolo inicial da gramática

Análise sintática ascendente

- A partir da gramática

<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>

- Reconhecer a cadeia:

[a;a]

Haveria outras opções de *handles*?

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a
$[L;S	]$	Reduz L → L;S
$[L	]$	Empilha]
$[L]	$	Reduz S → [L]
$S	$	**sucesso**

Análise sintática ascendente

- A partir da gramática

```
<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>
```

- Reconhecer a cadeia:

[a;a]

Haveria outras opções de *handles*?

- L → S
  - O que aconteceria?

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a
$[L;S	]$	Reduz L → L;S
$[L	]$	Empilha]
$[L]	$	Reduz S → [L]
$S	$	**sucesso**

Análise sintática ascendente

- A partir da gramática

```
<S> ::= [<L>] | a
<L> ::= <L>;<S> | <S>
```

- Reconhecer a cadeia:

[a;a]

Haveria outras opções de *handles*?

- L → S
  - O que aconteceria?
    - **$[L;L ERRO:** cadeia não é reconhecida

Pilha	Cadeia	Regra
$	[a;a]$	
$	[a;a]$	Empilha [
$[	a;a]$	Empilha a
$[a	;a]$	Reduz S → a
$[S	;a]$	Reduz L → S
$[L	;a]$	Empilha ;
$[L;	a]$	Empilha a
$[L;a	]$	Reduz S → a
$[L;S	]$	Reduz L → L;S
$[L	]$	Empilha]
$[L]	$	Reduz S → [L]
$S	$	**sucesso**

Análise sintática ascendente

- Operações durante a análise
  - Empilha: coloca-se no topo da pilha o primeiro símbolo da cadeia de entrada
  - Reduz: substitui-se o lado direito do *handle* pelo seu lado esquerdo
  - Aceita: a cadeia de entrada é reconhecida
  - Erro: a cadeia de entrada não é reconhecida

Análise sintática ascendente
Shift-Reduce

- Exercício:

1. Dada a gramática, mostre o reconhecimento para as entradas id+id, id+id*id e id+(id*id)

Processo de reconhecimento

1. Empilhar símbolos da cadeia de entrada
2. Quando um lado direito apropriado de uma produção aparece, ele é reduzido (substituído) pelo lado esquerdo da produção
3. Se a análise tiver sucesso, esse processo ocorre até que os símbolos da cadeia de entrada sejam todos consumidos e a pilha fique apenas com o símbolo inicial da gramática

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow id$

Processo de análise

Pilha	Cadeia	Regra
$	$	

Análise sintática ascendente

- *Bottom-up*, ascendente ou redutiva
  - Analisadores de precedência de operadores
  - Analisadores LR
    - SLR: *Simple* LR
    - LR Canônico
    - *Look Ahead* LR: LALR

Análise sintática ascendente
precedência de operadores

- Simples e eficiente
- Aplicada, principalmente, para o reconhecimento de expressões
- Subclasse de gramáticas
  - Gramáticas de (precedência de) operadores
    1. Não há símbolos não-terminais adjacentes
    2. Não há produções que derivam a cadeia nula

Análise sintática ascendente
precedência de operadores

- Exemplo: a gramática abaixo não é de precedência de operadores – três não-terminais consecutivos do lado direito

```
<E> ::= <E><O><E> | (<E>) | id
<O> ::= + | -
```

- Transformando-a em gramática de operadores:

```
<E> ::= <E>+<E> | <E>-<E> | (<E>) | id
```

Análise sintática ascendente
precedência de operadores

- Para identificar os *handles* (substituições), utilizam-se relações de precedência existentes entre os símbolos terminais (operandos e operadores) em uma tabela sintática (ou de precedência)
  - **<**, **>** e **=**

1. Relações de precedência ➔ Considere os terminais a e b
   - 1.1. a**<**b significa que a tem precedência menor do que b
   - 1.2. a**=**b significa que a e b têm a mesma precedência
   - 1.3. a**>**b significa que a tem precedência maior do que b

2. Durante a análise ascendente, na **pilha**:
   - 2.1. **<** identifica o limite esquerdo do lado direito do *handle*
   - 2.2. **=** indica que os terminais envolvidos pertencem ao mesmo *handle*
   - 2.3. **>** identifica o limite direito do lado direito do *handle*

Análise sintática ascendente
precedência de operadores

- Tabela sintática ➔ usando precedência de operadores
  - Matriz quadrada que relaciona todos os terminais da gramática e o símbolo delimitador utilizado (**$** ou λ ou ε)
    - Primeira linha da tabela: terminais da cadeia sendo analisada
    - Primeira coluna da tabela: terminais do topo da pilha

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	ok

→ cadeia

↓pilha

Análise sintática ascendente
precedência de operadores

- Regras para o uso da tabela sintática
- Seja a o terminal mais ao topo da pilha (os não-terminais são ignorados) e b o primeiro terminal da cadeia sendo analisada
  1. Se a<b ou a=b, então empilha b
  2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo

    - O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
      - **Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos**

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ **Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos**

❑ Tabela sintática ➔ usando precedência de operadores

❑ Ex: **id + id * id**

<E> ::= <E>+<E> | <E>*<E> | id

Cadeia a ser analisada
id + id * id$

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	ok

→ cadeia

↓pilha

1. $ na pilha com id na entrada resulta em < ➔ **$<**
2. Empilha id ➔ **$<id**
3. id na pilha com + na entrada resulta em > ➔ **$<id>**
4. Empilha + ➔ **$<id>+**
5. Na pilha tem + e na entrada id ➔ **$<id>+<**
6. Empilha id ➔ **$<id>+ <id**
7. id na pilha com * na entrada resulta em > ➔ **$<id>+ <id>**
8. Empilha * ➔ **$<id>+ <id>***
9. Na pilha tem * e na entrada id resulta em < ➔ **$<id>+ <id>*<**
10. Empilha id ➔ **$<id>+ <id>*<id**
11. Na pilha tem id e na entrada $ ➔ **$<id>+ <id>*<id>$**

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ **Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos**

❑ Tabela sintática ➔ usando precedência de operadores

❑ Ex: **id + id * id** **<E> ::= <E>+<E> | <E>*<E> | id**

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	ok

↓pilha

11. Na pilha tem id e na entrada $ ➔ **<id>+ <id>*<id>$**
12. Reduz id para E, segundo a gramática ➔ **$ + <id>*<id>$**
13. Reduz id para E, segundo a gramática ➔ **$ + *<id>$**
14. Reduz id para E, segundo a gramática ➔ **$ + * $**
15. Após a remoção dos não terminais ➔ **$ + * $**

Agora inserindo as relações de precedência segundo a tabela

16. Na pilha $ e na entrada + ➔ **$ <**
17. Empilha + ➔ **$< +**
18. Na pilha + e na entrada * ➔ **$ < + <**
19. Empilha * ➔ **$ < + < ***
20. Na pilha * e na entrada $ ➔ **$ < + < * >**
21. Indicando que o *handle* selecionado primeiramente é **E * E**

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Tabela sintática ➔ usando precedência de operadores

❑ Ex: **id + id * id** **<E> ::= <E>+<E> | <E>*<E> | id**

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	ok

↓pilha

11. Na pilha tem id e na entrada $ ➔ **<id>+ <id>*<id>$**
12. Reduz id para E, segundo a gramática ➔ **$ + <id>*<id>$**
13. Reduz id para E, segundo a gramática ➔ **$ + *<id>$**
14. Reduz id para E, segundo a gramática ➔ **$ + * $**
15. Após a remoção dos não terminais ➔ **$ + * $**

Agora inserindo as relações de precedência segundo a tabela

16. Na pilha $ e na entrada + ➔ **$ <**
17. Empilha + ➔ **$< +**
18. Na pilha + e na entrada * ➔ **$ < + <**
19. Empilha * ➔ **$ < + < ***
20. Na pilha * e na entrada $ ➔ **$ < + < * >**
21. Indicando que o *handle* selecionado primeiramente é **E * E**

Expressão id + id * id

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ *Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos*

❑ Tabela sintática ➔ usando precedência de operadores

❑ Ex: **id + id * id**

<E> ::= <E>+<E> | <E>*<E> | id

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	ok

↓pilha

pilha	cadeia	regra
$	id+id*id$	< empilha id
$<id	+id*id$	> reduz E-->id
$	+id*id$	< empilha +
$<+	id*id$	< empilha id
$<+<id	*id$	reduz
$<+	*id$	< empilha *
$<+ <*	id$	< empilha id
$<+ <* < id	$	> reduz
$<+ <*	$	> reduz
$<+	$	> reduz
$	$	aceitou

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

id&id/id

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha
$<&<id>	/id$	Reduz

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha
$<&<id>	/id$	Reduz
$<&>	/id$	Reduz

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha
$<&<id>	/id$	Reduz
$<&>	/id$	Reduz
$<	/id$	Empilha

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha
$<&<id>	/id$	Reduz
$<&>	/id$	Reduz
$<	/id$	Empilha
$</<	id$	Empilha

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha
$<&<id>	/id$	Reduz
$<&>	/id$	Reduz
$<	/id$	Empilha
$</<	id$	Empilha
$</<id>	$	Reduz

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha
$<&<id>	/id$	Reduz
$<&>	/id$	Reduz
$<	/id$	Empilha
$</<	id$	Empilha
$</<id>	$	Reduz
$</>	$	Reduz

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ Exemplo: expressões lógicas

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

→ cadeia

↓pilha

Pilha	Cadeia	Regra
$	id&id/id$	
$<	id&id/id$	Empilha
$<id>	&id/id$	Reduz
$<	&id/id$	Empilha
$<&<	id/id$	Empilha
$<&<id>	/id$	Reduz
$<&>	/id$	Reduz
$<	/id$	Empilha
$</<	id$	Empilha
$</<id>	$	Reduz
$</>	$	Reduz
$E	$	**Sucesso**

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos

❑ **Exercício**: reconheça a expressão

(id)

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

Pilha	Cadeia	Regra
$	(id)$	

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ **Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos**

❑ **Exercício**: resposta

(id)

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

Pilha	Cadeia	Regra
$	(id)$	
$<	(id)$	Empilha
$<(	id)$	Reduz
$<(<	id)$	Empilha
$<(<id	)$	Reduz
$<(<id>	)$	Reduz
$<(	)$	Empilha
$<(=)	$	Reduz
$<( )>	$	Reduz
$E	$	**Aceito**

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ **Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos**

❑ **Exercício**: reconheça a expressão

(id/id)&id

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

Pilha	Cadeia	Regra
$	(id/id)&id$	

1. Se a<b ou a=b, então empilha b **Obs: a→topo da pilha e b→terminal em análise**
2. Se a>b, então procura o lado direito do *handle* na pilha e o substitui pelo seu lado esquerdo
   - ❑ O lado direito do *handle* estará delimitado na pilha pelos símbolos < e >
     - ❑ **Os não terminais não precisam aparecer, mas se deve saber que foram produzidos e que seus derivados correspondentes foram consumidos**

❑ **Exercício**: resposta

(id/id)&id

<E> ::= <E>/<T> | <T>
<T> ::= <T>&<F> | <F>
<F> ::= (<E>) | id

	id	/	&	(	)	$
id		>	>		>	>
/	<	>	<	<	>	>
&	<	>	>	<	>	>
(	<	<	<	<	=	
)		>	>		>	>
$	<	<	<	<		

pilha	cadeia	regra
$	(id/id)&id$	< empilha (
$<(	id/id)&id$	< empilha id
$<(<id	/id)&id$	reduz F->id
$<(F	/id)&id$	reduz T->F
$<(T	/id)&id$	reduz E->T
$<(E	/id)&id$	reduz E->T
$<(E </	id)&id$	<empilha /
$<(E </<id	)&id$	<empilha id
$<(E </<F	)&id$	reduz F->id
$<(E </<T	)&id$	reduz T->F
$<(E	)&id$	> reduz E--> E/T
$<(=E)	&id$	= empilha
$<(=F)	&id$	> reduz F--> (E)
$T	&id$	> reduz T--> F
$T <&	id$	<empilha &
$T <&<id	$	<empilha id
$T <&F	$	reduz F->id
$T <&F	$	reduz T->T&F
$T	$	> reduz E--> T
$E	$	