Universidade de Santa Cruz do Sul – UNISC
Departamento de informática

COMPILADORES

Análise sintática

Parte 04

Prof. Geovane Griesang
geovanegriesang@unisc.br

Análise sintática

Data	Conteúdo
23/09/2013	3. Análise Sintática: 3.1 analisadores ascendentes e descendentes;
30/09/2013	3. Análise Sintática: 3.2 análise preditiva;
07/10/2013	3. Análise Sintática: 3.3 análise de precedência de operadores;
14/10/2013	3. Análise Sintática: 3.4 análise LR; → Próxima aula
21/10/2013	3. Análise Sintática: 3.5 recuperação de erros;
28/10/2013	3. Análise Sintática: 3.6 geradores de analisadores sintáticos;
04/11/2013	Prova Individual e sem consulta 2.

Análise sintática

Legenda:

$\sum$ = sigma (somatório)
δ = delta
ε = épsilon
λ = lambda
α = alfa
β = beta
γ = gamma
ξ = xi

Análise sintática

Análise ascendente

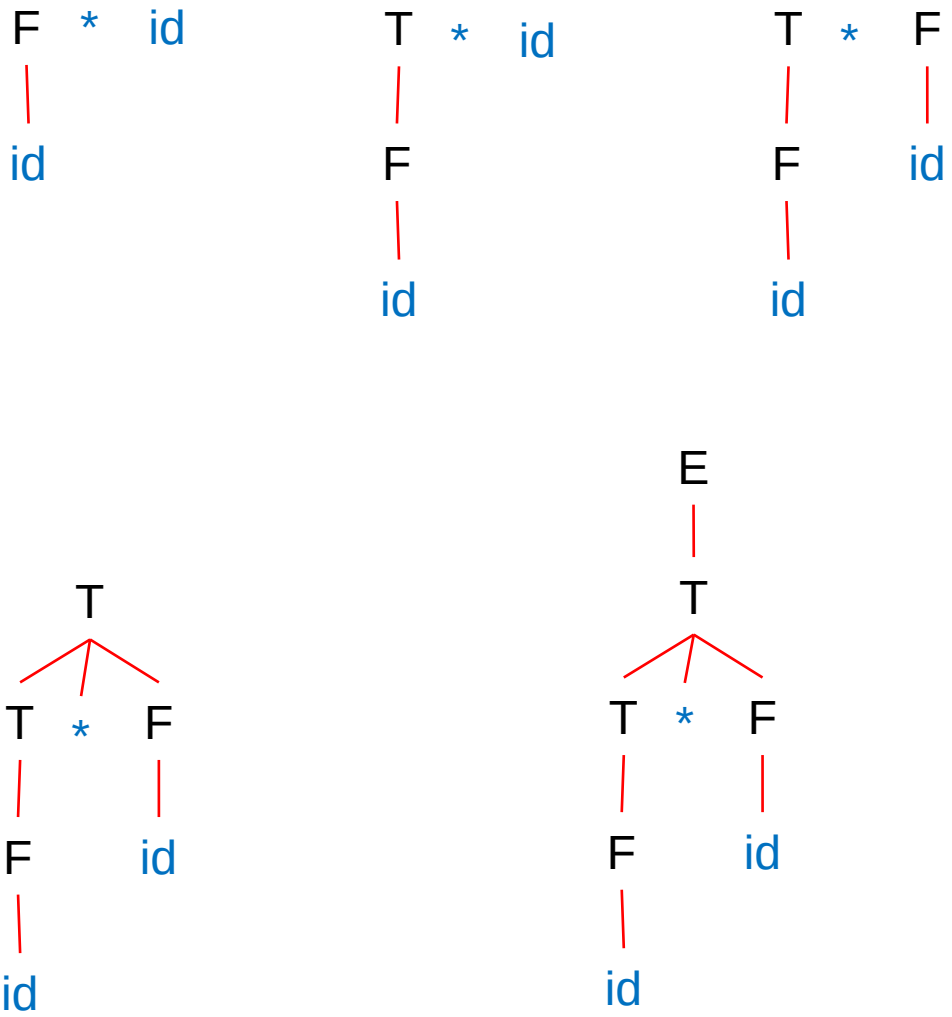
A análise sintática ascendente corresponde à construção de uma árvore de derivação para uma cadeia de entrada a partir das folhas (a parte de baixo) em direção a raiz (o topo) da árvore.

Embora a árvore de derivação seja usada para descrever os métodos de análise, um *front-end* pode executar uma tradução diretamente, portanto, na prática, ela nunca é efetivamente construída.

Análise sintática

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

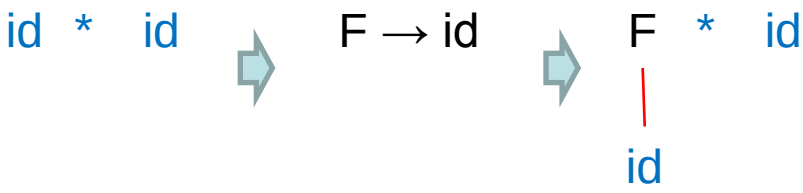
Entrada: id * id



Análise sintática

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

Análise ascendente



Reduções

Podemos pensar na análise ascendente como processo de "reduzir" uma cadeia *w* para o símbolo inicial da gramática.

Em cada passo da *redução*, uma subcadeia específica, casando com o lado direito de uma produção, é substituída pelo não-terminal na cabeça dessa produção.

As principais decisões relacionadas com a análise ascendente em cada passo do reconhecimento são: determinar quando reduzir e determinar a redução a ser usada para que a análise prossiga.

Entrada: id * id

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

Análise sintática

id * id

F → id, F * id

T → F, T * id

Análise ascendente

Reduções - Exemplo

A derivação segue a seguinte sequencia de cadeias:

id * id, F * id, T * id, T * F, T, E

A sequencia começa com a cadeia de entrada: id * id.

A primeira redução produz F * id, reduzindo o id mais a esquerda para F, usando a produção F → id.

A segunda redução produz T * id, reduzindo F para T.

Entrada: id * id

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

Análise sintática

id * id

F → id, F * id

T → F, T * id

E → T, E * id ou F → id, T * F

Análise ascendente

Reduções - Exemplo

A derivação segue a seguinte sequencia de cadeias:

id * id, F * id, T * id, T * F, T, E

Neste ponto temos duas opções de redução:

podemos reduzir a cadeia T para E, segundo a produção E → T, e a cadeia consistindo no segundo id, que representa o lado direito de F → id.

Escolhemos a segunda opção e reduzimos id para F, produzindo a cadeia T * F.

Análise sintática

Entrada: id * id

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

Análise ascendente

Reduções - Exemplo

		id * id
		F → id, F * id
		T → F, T * id
E → T, E * id	ou	F → id, T * F
		T → T * F, T
		E → T, E

A derivação segue a seguinte sequencia de cadeias:

id * id, F * id, T * id, T * F, T, E

Essa cadeia é então reduzida para T.

A análise termina com a redução de T para o símbolo inicial E.

Análise sintática

Análise ascendente

Reduções

Por definição, uma redução é o inverso de um passo em uma derivação (lembre-se de que, em uma derivação, um não-terminal em uma forma sentencial é substituído pelo lado direito de uma de suas produções).

O objetivo do método de análise ascendente é, portanto, construir uma derivação ao reverso.

Então, essa derivação é, na verdade, uma derivação mais à direita.

$$E \Rightarrow T \Rightarrow T * F \Rightarrow T * id \Rightarrow F * id \Rightarrow id * id$$

Análise sintática

Análise ascendente

Poda do *Handle*

A análise ascendente ao ler a entrada, da esquerda para a direita, constrói uma derivação mais à direita ao inverso.

Informalmente, um "*handle*" de uma cadeia de símbolos é uma subcadeia que casa com o corpo da produção, e cuja redução para o não-terminal do lado esquerdo representa um passo da derivação à direita ao inverso.

Informalmente, um *Handle* pode ser definido como: lado direito de uma produção que, se for reduzido, seguido de outras reduções, vai levar para o símbolo inicial.

Entrada: id * id

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

Análise sintática

Análise ascendente

Poda do *Handle:* ex.: *Handles* durante a análise de $id_1 * id_2$:

Forma sentencial à direita	*Handle*	Produção de redução
$id_1 * id_2$	id_1	F → id
$F * id_2$	F	T → F
$T * id_2$	id_2	F → id
T * F	T * F	E → T * F

Embora T seja o lado direito da produção E → T, o símbolo T não é um *handle* na forma sentencial $T * id_2$.

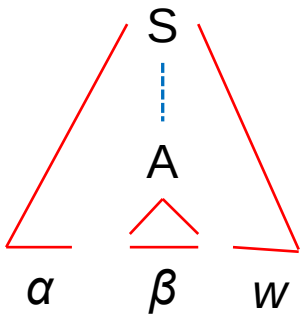
Se T fosse realmente substituído por E, obteríamos a cadeia $E * id_2$, a qual não pode ser derivada a partir do símbolo inicial E.

Análise sintática

Análise ascendente

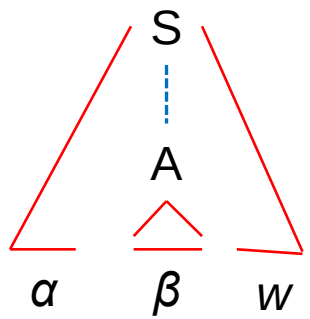
Poda do *Handle*

Formalmente, se $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha Aw \Rightarrow \alpha\beta w$, então a produção $A \rightarrow \beta$ na posição seguindo α é um *handle* de $\alpha\beta w$.



Alternativamente, um *handle* de uma forma sentencial mais à direita é uma produção $A \rightarrow \beta$ e uma posição em y onde a cadeia β pode ser localizada, de modo que a substituição de β *nessa posição por* A produz forma sentencial mais à direita anterior em uma derivação mais a direita de y.

Análise sintática



Análise ascendente

Poda do *Handle*

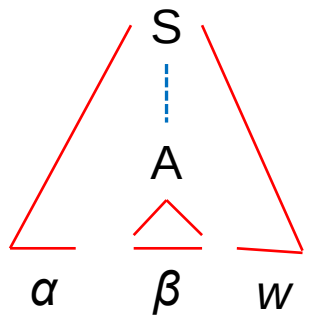
A cadeia w à direita de *handle* deve conter apenas símbolos terminais.

Por conveniência, vamos nos referir ao lado direito de *β* ao invés de A→ *β* com um *handle*.

Dizemos “um *handle*”, em vez de “o *handle*”, pois a gramática pode ser ambígua, com mais de uma derivação a direita para *αβw.*

Se uma gramática não for ambígua, então cada forma sentencial a direita da gramática à direita tem exatamente um *handle*.

Análise sintática



Análise ascendente

Poda do *Handle*

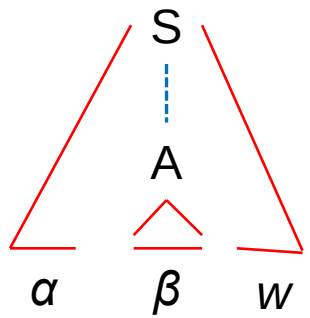
Uma derivação mais à direita ao reverso pode ser obtida pela "poda do *handle*".

Começamos com a cadeia de terminais w a ser analisada.

Se w for a sentença da gramática dada, então considere $w = \gamma_n$, onde γ_n é a enésima forma sentencial mais à direita de alguma derivação mais à direita ainda não conhecida.

$$S = \gamma_0 \underset{rm}{\Rightarrow} \gamma_1 \underset{rm}{\Rightarrow} \gamma_2 \underset{rm}{\Rightarrow} \gamma_{n-1} \underset{rm}{\Rightarrow} \gamma_n = w$$

Análise sintática



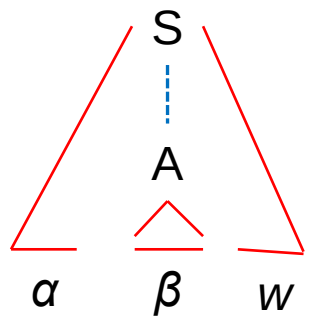
Análise ascendente

Poda do *Handle*

Para reconstruir essa derivação em ordem reversa, localizamos o *handle* β_n em y_n e substituímos β_n pelo lado esquerdo de alguma produção relevante $A_n \rightarrow \beta_n$ p/ obter a forma sentencial anterior mais à direita de y_{n-1}.

Métodos para o reconhecimento dos *handles* serão estudados nas próximas aulas.

Análise sintática



Análise ascendente

Poda do *Handle*

Repetimos esse processo, ou seja, localizamos o *handle* β_{n-1} em y_{n-1} e reduzimos esse *handle* para obter a forma mais sentencial mais à direita y_{n-2}.

Se continuarmos esse processo e conseguirmos alcançar o símbolo inicial da gramática, paramos e anunciamos o sucesso no reconhecimento.

O reverso da sequencia de produções usadas nas reduções é uma derivação mais à direita para a cadeia da entrada.

Análise sintática

Análise ascendente

Analisador sintático *Shift-reduce*

O analisador sintático *shift-reduce* é uma forma de análise ascendente em que uma pilha contém símbolos da gramática e um *buffer* de entrada contém o restante da cadeia a ser reconhecida sintaticamente.

O *handle* sempre aparece no topo da pilha, imediatamente antes de ser identificado.

Usa-se o símbolo $ para marcar o fundo da pilha e também o extremo direito da entrada.

Análise sintática

Análise ascendente

Analisador sintático *Shift-reduce*

Convenção adotadas para a análise ascendente:

Mostraremos o topo da pilha à direita, em vez de à esquerda, como fizemos para a análise descendente.

Inicialmente, a pilha está vazia, e a cadeia *w* representa a entrada, da seguinte forma:

Pilha	Entrada
$	w$

Análise sintática

Análise ascendente

Analisador sintático *Shift-reduce*

Durante uma escansão da esquerda para direita da cadeia de entrada, o analisador sintático transfere zero ou mais símbolos da entrada para a pilha, até que uma cadeia β de símbolos da gramática presente no topo possa ser reduzida para o lado esquerdo da produção apropriada.

O analisador repete esse ciclo até detectar um erro na entrada ou até que a pilha contenha apenas o símbolo inicial da gramática e a entrada esteja vazia:

Pilha	Entrada
$S	$

Entrada: id * id

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

Análise sintática

Análise ascendente

Analisador sintático *Shift-reduce*

Ao entrar nessa configuração, o analisador para e anuncia sucesso no reconhecimento da cadeia de entrada $id_1 * id_2$:

Pilha	Entrada	Ação
$	$id_1 * id_2$ $	Transfere
$ id_1	* id_2 $	Reduz segundo F → id
$ F	* id_2 $	Reduz segundo T → F
$ T	* id_2 $	Transfere
$ T *	id_2 $	Transfere
$ T * id_2	$	Reduz segundo F → id
$ T * F	$	Reduz segundo T → T * F
$ T	$	Reduz segundo E → T
$ E	$	Aceita

Análise sintática

Análise ascendente

Analisador sintático *Shift-reduce*

Embora as principais operações sejam *shift* e *reduce*, existem quatro operações possíveis que um analisador *shift-reduce* pode realizar:

1. *Shift* (do inglês “empilha e avança”): Transfere o próximo símbolo da entrada para o topo da pilha.

2. *Reduce*: O extremo direito da cadeia a ser reduzida deve estar no topo da pilha.

   Localize o extremo esquerdo da cadeia no interior da pilha e decida por qual não-terminal esta cadeia será substituída.

Análise sintática

Análise ascendente

Analisador sintático *Shift-reduce*

3. *Accept*: Anuncia o término bem sucedido da análise.

4. *Error*: Ao descobrir um erro de sintaxe chame uma determinada rotina de recuperação de erro.

O uso de uma pilha durante a análise *shift-reduce* é justificado por um fato importante: o *handle* sempre aparece do topo para baixo na pilha do reconhecedor, nunca em seu interior.

Esse fato pode ser mostrado considerando-se as formas possíveis de dois passos sucessivos em qualquer derivação mais à direita.

Casos possíveis nó próximo *slide*.

Análise sintática

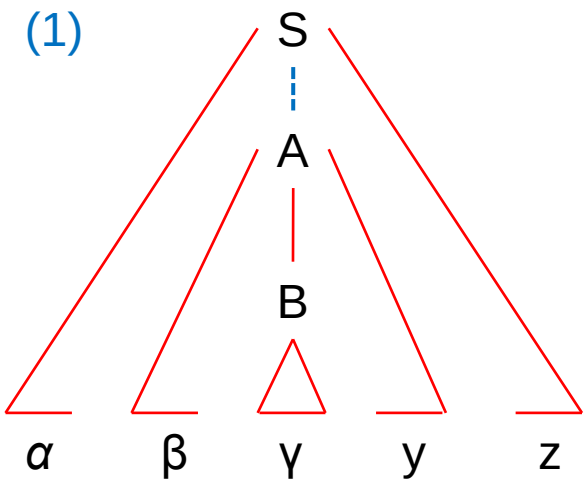
Análise ascendente

Analisador sintático *Shift-reduce*

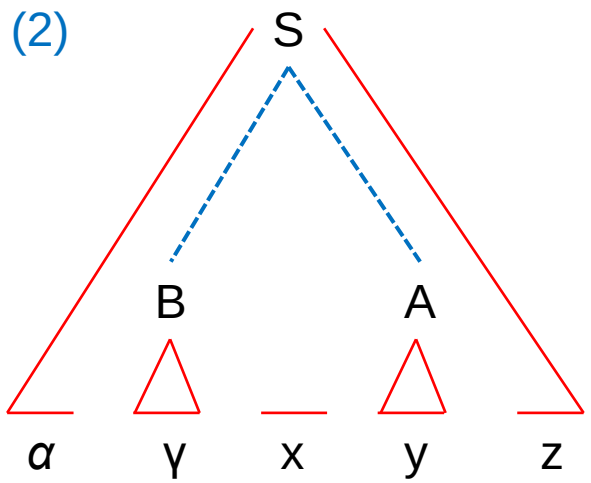
No caso (1), A é substituído por βBy, e então o não-terminal mais à direita B no corpo βBy é substituído por γ.

No caso (2), novamente A é derivado primeiro, mas, dessa vez, o corpo é uma cadeia γ contendo só terminais. O próximo não-terminal mais à direita B estará em algum ponto à esq. de y.

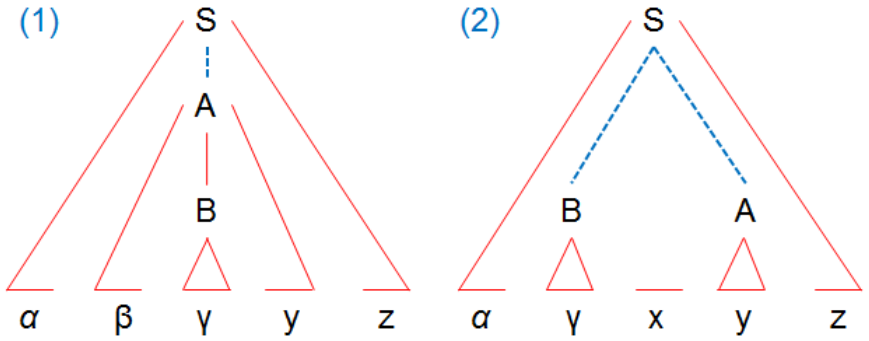
(1)



(2)



Análise sintática



Análise ascendente

Analisador sintático *Shift-reduce*

No caso (1), A é substituído por βBy, e então o não-terminal mais à direita B no corpo βBy é substituído por γ.

No caso (2), novamente A é derivado primeiro, mas, dessa vez, o corpo é uma cadeia γ contendo só terminais. O próximo não-terminal mais à direita B estará em algum ponto à esq. de y.

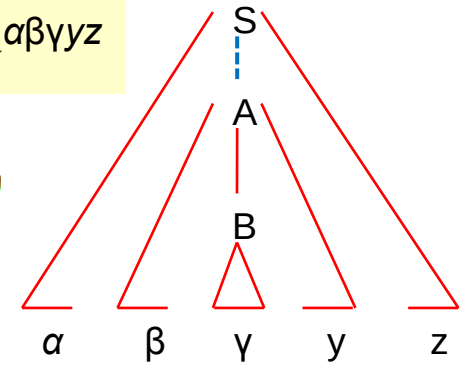
Em outras palavras:

$$(1)\ S \overset{*}{\underset{rm}{\Rightarrow}} \alpha Az \underset{rm}{\Rightarrow} \alpha\beta Byz \underset{rm}{\Rightarrow} \alpha\beta\gamma yz$$

$$(2)\ S \overset{*}{\underset{rm}{\Rightarrow}} \alpha BxAz \underset{rm}{\Rightarrow} \alpha Bxyz \underset{rm}{\Rightarrow} \alpha\gamma xyz$$

Análise sintática

(1) $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha Az \underset{rm}{\Rightarrow} \alpha\beta Byz \underset{rm}{\Rightarrow} \alpha\beta\gamma yz$



Análise ascendente

Analisador sintático *Shift-reduce*

Considere o caso (1) ao inverso, onde o analisador *shift-reduce* acabou de atingir a configuração.

Pilha	Entrada
$αβγ	yz$

O analisador reduz *handle* γ para B para atingir a configuração.

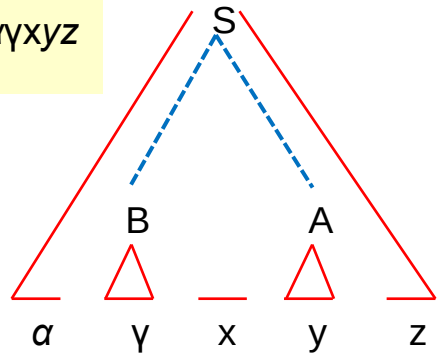
$αβB	yz$

O analisador sintático agora pode transferir a cadeia y p/ a pilha em uma sequencia de zero ou mais movimentos de transferência para atingir a configuração. Com o handle βBy no topo da pilha, ele é reduzido para A.

$αβBy	z$

Análise sintática

(2) $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha Bx Az \underset{rm}{\Rightarrow} \alpha Bxyz \underset{rm}{\Rightarrow} \alpha\gamma xyz$



Análise ascendente

Analisador sintático *Shift-reduce*

Pilha	Entrada
$αγ	xyz$

Considere o caso (2), o *handle* está no topo da pilha. Após reduzi-lo para B, o analisador pode transferir a cadeia xy p/ deixar o próximo *handle* y no topo da pilha, pronto para ser reduzido para A.

$αBxy	z$

Em ambos os casos, após uma redução, o analisador sintático teve de transferir zero ou mais símbolos a fim de deixar o próximo *handle* no topo da pilha, mas nunca teve de verificar o interior da pilha para encontrar o *handle*.

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

Existem gramáticas livres de contexto para as quais a análise *shift-reduce* não pode ser aplicada.

Todo analisador *shift-reduce* para tais gramáticas pode alcançar uma configuração na qual, conhecendo todo conteúdo da pilha e o próximo símbolo da entrada, o analisador:

- não é capaz de decidir se amplia e avança ou reduz, causando o *conflito shift-reduce*,
- ou não consegue decidir qual das várias reduções disponíveis deverá ser aplicada, caracterizando um *conflito reduz/reduz*.

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

Tecnicamente, essas gramáticas não pertencem as gramáticas LR(k), assim, nos referimos a essas gramáticas como não-LR.

O k em LR(k) refere-se ao número de símbolos à frente na entrada ainda não lidos, conhecidos como *lookaheads*.

As gramáticas usadas na compilação usualmente pertencem a gramática LR(1), e na prática apenas um símbolo à frente na entrada é suficiente.

As gramáticas LR serão estudadas nas próximas aulas.

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

Uma gramática LR nunca pode ser ambígua. Exemplo: gramática do *else*.

```
stmt → if expr then stmt
     |  if expr then stmt else stmt
     |  other
```

Se um analisador *shift-reduce* em um momento tiver a configuração:

Pilha	Entrada
... if expr then stmt	else ... $

Não há como saber se *if expr them stmt* é um *handle*, não importa o que aparece abaixo dele na pilha.

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

```
stmt → if expr then stmt
     |  if expr then stmt else stmt
     |  other
```

Pilha	Entrada
... if expr then stmt	else ... $

Essa configuração da pilha ilustra um conflito *shift-reduce*.

Dependendo do que segue *else* na entrada, pode ser correto reduzir *if expr then stmt* para *stmt*, ou pode ser correto transferir *else* para pilha e então procurar outro *stmt* para completar a aternativa *if expr then stmt else stmt*.

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

stmt → if expr then stmt
| if expr then stmt else stmt
| other

Pilha	Entrada
... if expr then stmt	else ... $

Outro cenário comum para conflitos ocorre quando sabemos que temos um *handle*, mas o conteúdo da pilha e o próximo símbolo da entrada são insuficientes para determinar qual produção deve ser usada em uma redução.

O exemplo do próximo *slide* ilustra essa situação.

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

Ex.: Suponha que o analisador léxico retorne id como o nome do *token* para todos os nomes, independente do seu tipo.

Suponha também que, nossa linguagem invoque procedimentos dando seus nomes, com parâmetros entre parênteses, e que os arranjos sejam referenciados com a mesma sintaxe.

Como a tradução dos índices referenciados em arranjos e parâmetros nas chamadas dos procedimentos são diferentes, usa-se produções diferentes para gerar as listas de parâmetros atuais e para os índices.

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: Portanto, nossa gramática pode ter as seguintes produções:

(1)	stmt	→	id(parameter_list)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(expr_list)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: um comando começando com p(i, j) é representado internamente pela sequencia de *tokens* id(id, id) durante a análise sintática.

Após transferir os três primeiros *tokens* para pilha, um *shift-reduce* estaria na configuração:

Pilha	Entrada
... id (id	, id) ...

(1)	stmt	→	id(parameter_list)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(expr_list)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: é evidente que o id no topo da pilha precisa ser reduzido, mas por qual produção?

A escolha correta é a produção (5) se for um procedimento, mas é (7) se for um arranjo.

Pilha	Entrada
... id (id	, id) ...

(1)	stmt	→	id(parameter_list)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(expr_list)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: uma simples inspeção na pilha não é suficiente para nos dizer qual produção escolher.

É necessário usar as informações obtidas na tabela de símbolos obtida a partir da declaração de p.

Pilha	Entrada
... id (id	, id) ...

(1)	stmt	→	id(parameter_list)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(expr_list)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: uma solução é substituir o *token* id da produção (1) para procid e usar um analisador léxico mais sofisticado, que retorne o nome do *token* procid quando ele reconhecer um lexema que seja o nome de um procedimento.

Pilha	Entrada
... id (id	, id) ...

(1)	stmt	→	**procid**(parameter_list)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(expr_list)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: Para fazer isto, é preciso que o analisador léxico consulte a tabela de símbolo antes de retornar um *token*. Se fizermos essa modificação, então, no procedimento p(i, j), o analisador estaria na configuração:

Pilha	Entrada
... procid (id	, id) ...

... ou na configuração anterior:

Pilha	Entrada
... id (id	, id) ...

(1)	stmt	→	**procid**(parameter_list)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(expr_list)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: No segundo caso, escolhemos a produção (5).

Pilha	Entrada
... procid (id	, id) ...

(1)	stmt	→	**procid**(parameter_list)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(expr_list)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

Análise sintática

Análise ascendente

Conflito durante a análise sintática *shift-reduce*

... Ex.: Pode-se observar que o terceiro símbolo a partir do topo da pilha para baixo determina a redução a ser feita, embora não esteja envolvido na redução.

(1)	stmt	→	procid(**parameter_list**)
(2)	stmt	→	expr := expr
(3)	parameter_list	→	parameter_list, parameter
(4)	parameter_list	→	parameter
(5)	parameter	→	id
(6)	expr	→	id(**expr_list**)
(7)	expr	→	id
(8)	expr_list	→	expr_list, expr
(9)	expr_list	→	expr

A análise sintática *shift-reduce* pode utilizar informações mais abaixo na pilha para auxiliar o reconhecimento da cadeia de entrada.

Análise sintática

Análise ascendente

Análise LR: *Left to Right with Rightmost derivation*

Atualmente, o tipo mais usado de analisadores sintático ascendentes é baseado em um conceito chamado LR(K).

L representa a escanção de entrada da esquerda para direita.

R representa a construções mais à direita ao reverso.

k representa os símbolos a frente do fluxo de entrada que auxiliam nas decisões de análise.

COMPILADORES

Obrigado!!

Prof. Geovane Griesang
geovanegriesang@unisc.br