

2.3 - Análise Sintática

- ♦ Vamos supor que a sintaxe das linguagens que iremos estudar está totalmente descrita por uma gramática livre de contexto. Isto é uma simplificação. Na realidade, as linguagens de programação, mesmo que descritas por GLC, apresentam vários aspectos não-livres de contexto. Essa simplificação, no entanto, é muito conveniente, dada a simplicidade dos reconhecedores para as LLC - os autômatos com pilha (PDA).
- ♦ Para ilustrar, considere que o conjunto de programas válidos de uma linguagem de programação P é o conjunto de sentenças que:
 - a) satisfazem as condições sintáticas de P
 - b) satisfazem as condições semânticas de P

Como, em geral, a sintaxe de uma linguagem de programação é escrita em BNF, não existe dúvida de que o conjunto das cadeias que satisfazem (a) é livre de contexto. Entretanto, como a semântica é estabelecida em linguagem natural, não é evidente se o conjunto das cadeias que satisfazem (a) e (b) é livre de contexto.

- ♦ Exemplo: Seja o conjunto $L = \{ wcw \mid w \in \Sigma^* \}$. L não é livre de contexto (lembrar que $\{ wcw^r \mid w \in \Sigma^*, w^r \text{ reverso de } w \}$ é livre de contexto). Entretanto, o problema de reconhecer L pode ser visto como o problema de verificar se os identificadores usados em um programa foram declarados, ou seja, o primeiro w em wcw representa a declaração de um identificador e o segundo w , o seu uso no programa. Portanto, se a linguagem P exige que os identificadores sejam declarados antes de serem usados, P não é LLC.

Análise Sintática

- Seja G uma GLC. Dizemos que uma sentença $w \in L(G)$ foi analisada sintaticamente quando conhecemos uma de (ou talvez todas) suas árvores de derivação.
- Relembrando:
 - $E \rightarrow E + T$
 - $E \rightarrow T$
 - $T \rightarrow T * F$
 - $T \rightarrow F$
 - $F \rightarrow (E)$
 - $F \rightarrow a$

Seja $w = a * (a + a)$

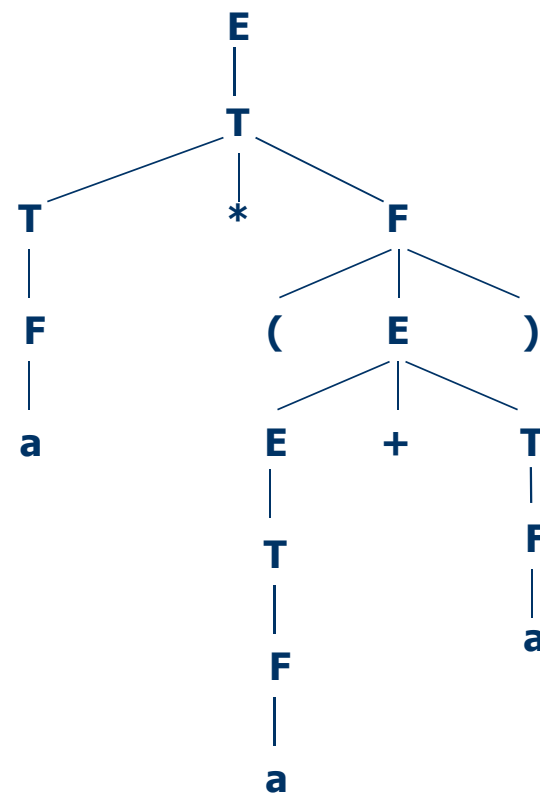
Análise sintática descendente (sequência de produções usadas numa derivação mais à esquerda):

2 3 4 6 5 1 2 4 6 4 6

Análise sintática ascendente (reverso de uma sequência de produções usadas numa derivação mais à direita):

6 4 6 4 2 6 4 1 5 3 2

ÁRVORE DE DERIVAÇÃO



Análise Sintática Descendente (“Top-down”)

- ♦ Como vimos, o nome da análise sintática “descendente” vem da idéia de tentar produzir a árvore de derivação para a cadeia de entrada, da raiz para (“descendo até”) as folhas.
- ♦ Comentamos também que existe forte evidência que as gramáticas livres de contexto determinísticas (LL(k)) sejam adequadas para especificar as características sintáticas de linguagens de programação.
- ♦ Alguns resultados já conhecidos:
 1. uma gramática recursiva à esquerda não pode ser LL(k), para qualquer k.
 2. uma gramática ambígua não pode ser LL(k), para qualquer k.
- ♦ Vamos admitir que a gramática que descreve a linguagem de programação é LL(1). E se a gramática não for LL(1)?

Existem algumas transformações que podem levar gramáticas não LL(1) em gramáticas LL(1) equivalentes:

1) Eliminação da recursão à esquerda

Sejam: $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n$ as A-produções diretamente recursivas à esquerda e $A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$ as A-produções não recursivas à esquerda. Este conjunto de produções gera:

$$(\beta_1 \mid \beta_2 \mid \dots \mid \beta_m)(\alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n)^*$$

Logo: podemos substituir as A-produções por:

$$\begin{aligned} A &\rightarrow \beta_1 Z \mid \beta_2 Z \mid \dots \mid \beta_m Z \\ Z &\rightarrow \alpha_1 Z \mid \alpha_2 Z \mid \dots \mid \alpha_n Z \mid \Lambda \end{aligned}$$

Análise Sintática Descendente (“Top-down”)

2. Fatoração

A fatoração é usada quando se tem:

$$X \Rightarrow \alpha \Rightarrow^* a\beta$$

$$X \Rightarrow \alpha' \Rightarrow^* a\beta'$$

Fatoração direta:

$X \rightarrow a\alpha_1 \mid a\alpha_2 \mid \dots \mid a\alpha_n$ pode ser transformada para:

$$X \rightarrow aZ$$

$$Z \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

Fatoração indireta: a fatoração é feita em vários passos.

Exemplo: $A \rightarrow Bc \mid De$
 $B \rightarrow bX \mid Z$
 $D \rightarrow bY$

Nesse caso:

$$A \Rightarrow Bc \Rightarrow bXc$$

$$A \Rightarrow De \Rightarrow bYe \quad (\text{não é LL(1) !})$$

Fazemos então: $A \rightarrow bXc \mid Zc \mid bYe$

e, em seguida: $A \rightarrow bW \mid Zc$

$$W \rightarrow Xc \mid Ye$$

Análise Sintática Descendente (“Top-down”)

Implementação de analisadores LL(1) : Descida recursiva

- Nessa técnica, para cada não-terminal da gramática escrevemos um procedimento e, ao invés de empilhar o lado direito da produção, usamos chamadas recursivas (ou seja, a pilha é implícita).

- Exemplo: Seja a gramática:

```
<comando> ::= <var> := <expr> |  
              IF <expr> THEN <comando> |  
              IF <expr> THEN <comando> ELSE <comando>  
  
<var> ::= i | i (<expr>)  
<expr> ::= <termo> | <expr> + <termo>  
<termo> ::= <fator> | <termo> * <fator>  
<fator> ::= <var> | (<expr>)
```

Eliminando a recursão à esquerda e fatorando, temos:

```
<comando> ::= <var> := <expr> |  
              IF <expr> THEN <comando> [ ELSE <comando> ]  
  
<var> ::= i [ (<expr>) ]  
<expr> ::= <termo> { + <termo> }  
<termo> ::= <fator> { * <fator> }  
<fator> ::= <var> | (<expr>)
```

↑
termos entre colchetes são
opcionais e entre chaves podem
ser repetidos 0 ou mais vezes.

Análise Sintática Descendente ("Top-down")

- Para a gramática assim transformada podemos escrever os seguintes procedimentos:

```
<comando> ::= <var> := <expr> |  
             IF <expr> THEN <comando> [ ELSE <comando> ]
```

```
procedure COMANDO;  
  if SIMB = "IF" then  
  begin  
    SCAN;  
    EXPR;  
    if SIMB ≠ "THEN" then ERRO  
  else  
  begin  
    SCAN;  
    COMANDO;  
    if SIMB = "ELSE" then  
    begin  
      SCAN;  
      COMANDO;  
    end;  
  end;  
end
```

```
  else  
  begin  
    VAR;  
    if SIMB ≠ ":=" then ERRO  
  else  
  begin  
    SCAN;  
    EXPR;  
  end;  
end;
```

Análise Sintática Descendente ("Top-down")

`<var> ::= i [(<expr>)]`

```
procedure VAR;  
  if SIMB ≠ "i" then ERRO  
  else  
    begin  
      SCAN;  
      if SIMB = "(" then  
        begin  
          SCAN;  
          EXPR;  
          if SIMB ≠ ")" then ERRO  
          else  
            SCAN;  
        end;  
      end;  
    end;  
end;
```

`<expr> ::= <termo> { + <termo> }`

```
procedure EXPR;  
begin  
  TERMO;  
  while SIMB = "+" do  
    begin  
      SCAN;  
      TERMO;  
    end;  
  end;  
end;
```

Análise Sintática Descendente (“Top-down”)

$\langle \text{termo} \rangle ::= \langle \text{fator} \rangle \{ * \langle \text{fator} \rangle \}$

```
procedure TERMO;  
begin  
  FATOR;  
  while SIMB = "*" do  
    begin  
      SCAN;  
      FATOR;  
    end;  
end;
```

$\langle \text{fator} \rangle ::= \langle \text{var} \rangle \mid (\langle \text{expr} \rangle)$

```
procedure FATOR;  
if SIMB = "(" then  
begin  
  SCAN;  
  EXPR;  
  if SIMB ≠ ")" then ERRO  
  else  
    SCAN;  
end  
else  
  VAR;
```

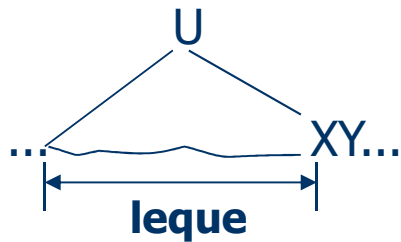

Análise Sintática Ascendente (“Bottom-up”)

- ♦ Como vimos, na análise sintática ascendente parte-se das folhas (isto é, dos próprios símbolos de entrada) e tenta-se construir a árvore sintática “subindo” até a raiz.
- ♦ Vimos também que o problema com o método “bottom-up” é encontrar um leque (“handle”) e uma vez encontrado, saber para qual não-terminal reduzi-lo. Vimos que isso pode ser feito de forma determinística para as gramáticas LR(k), com o uso de tabelas constituídas por uma função de ação de análise (que contém a informação necessária sobre o que deve ser feito: empilhar ou reduzir) e uma função de movimento (que contém a informação necessária para reconhecer que existe um leque no topo da pilha). Dependendo de k, a análise de gramáticas LR(k) envolve tabelas muito grandes.
- ♦ Por razões práticas, vamos considerar a análise sintática ascendente para uma certa classe de gramáticas denominadas gramáticas de precedência.

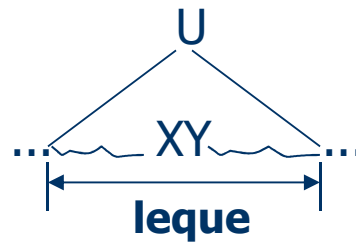
Sejam X e Y dois símbolos do vocabulário ($N \cup \Sigma$) de uma gramática G. Vamos supor que existe uma forma sentencial ...XY...

Então, em algum ponto da análise sintática ascendente, ou X, ou Y, ou ambos devem pertencer a algum leque. Temos então as seguintes possibilidades:

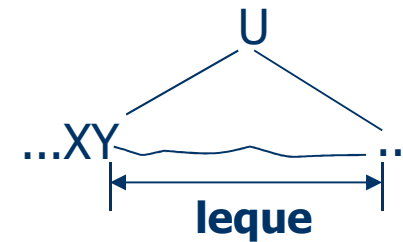
Gramáticas de precedência



(a)



(b)



(c)

- a) X é parte de um leque mas Y não é. Nesse caso, dizemos que X tem precedência sobre Y ($X \gg Y$) porque X deve ser reduzido primeiro.
 - b) X e Y estão ambos no mesmo leque. Nesse caso, dizemos que X e Y têm a mesma precedência ($X == Y$).
 - c) Y é parte de um leque mas X não é. Nesse caso, Y tem precedência sobre X ($X \ll Y$).
-
- ♦ Se não existir uma forma sentencial ...XY... então vamos dizer que não existe relação de precedência para o par ordenado (X,Y).

Gramáticas de precedência

- Exemplo: $S \rightarrow bAb$
 $A \rightarrow (B \mid a$
 $B \rightarrow Aa)$

Forma sentencial	bab	$b(Bb$	$b(Aa)b$
Árvore sintática	<pre> graph TD S --> b1[b] S --> A1[A] S --> b2[b] A1 --> a[a] </pre>	<pre> graph TD S --> b1[b] S --> A1[A] S --> b2[b] A1 --> LP["("] A1 --> B[B] </pre>	<pre> graph TD S --> b1[b] S --> A1[A] S --> b2[b] A1 --> LP["("] A1 --> B[B] B --> A2[A] B --> a[a] B --> RP[")"] </pre>
Leque	a	$(B$	$Aa)$
Relações de precedência	$b \ll a$ $a \gg b$	$b \ll ($ $(== B$ $B \gg b$	$(\ll A$ $A == a$ $a ==)$ $) \gg b$

Gramáticas de precedência

Formalmente:

- Definição: As relações de precedência de Wirth-Weber $<<, ==, >>$ para uma GLC $G = (N, \Sigma, P, S)$ são definidas sobre $(N \cup \Sigma)$ como:
 - a) $X << Y$ se existe $A \rightarrow \alpha XB\beta \in P$ tal que $B \Rightarrow^+ Y\gamma$
 - b) $X == Y$ se existe $A \rightarrow \alpha XY\beta \in P$
 - c) $X >> a$ se existe $A \rightarrow \alpha BY\beta \in P$ tal que $B \Rightarrow^+ \gamma X$ e $Y \Rightarrow^* a\delta$
(notar que $>>$ é definida em $(N \cup \Sigma) \times \Sigma$ porque o símbolo à direita de um leque em uma forma sentencial de uma derivação mais à direita é sempre um terminal)
- Definição: Uma GLC G é própria se não existe derivação da forma $A \Rightarrow^+ A$, não existem símbolos inúteis em G e G é Λ -livre (ou seja, a única Λ -produção é, possivelmente, $S \rightarrow \Lambda$, caso em que S não aparece no lado direito de qualquer produção)
- Definição: Uma GLC é univocamente invertível se não existirem duas produções com o mesmo lado direito.
- Definição: Uma GLC própria $G = (N, \Sigma, P, S)$ na qual existe, no máximo, uma relação de precedência de Wirth-Weber entre qualquer par de símbolos de $(N \cup \Sigma)$ é chamada gramática de precedência. Uma gramática de precedência univocamente invertível é chamada gramática de precedência simples.

Gramáticas de precedência

- Como as relações de precedência ajudam na análise sintática ascendente?
Seja \$ um símbolo não pertencente a $(N \cup \Sigma)$. Vamos fazer $\$ \ll X, X \gg \$$ para todo símbolo $X \in (N \cup \Sigma)$. O símbolo \$ será o delimitador de início e fim de uma forma sentencial.
- Teorema: Uma gramática de precedência é não ambígua. Além disso, o único leque de uma forma sentencial $S_1 S_2 \dots S_n$ qualquer é a subcadeia mais à esquerda $S_i \dots S_j$ tal que $S_{i-1} \ll S_i == S_{i+1} == \dots == S_j \gg S_{j+1}$
- Prova: Exercício!

Por esse teorema, o leque de uma forma sentencial pode ser localizado percorrendo a forma sentencial da esquerda para a direita até que seja encontrada a relação \gg . O extremo esquerdo do leque é localizado voltando atrás até encontrar a relação \ll .

Construção das relações de precedência

Sejam R e P relações binárias sobre um conjunto C. Então:

- a transposta da relação R - escrita como $\text{TRANS}(R)$ - é definida por:
$$a \text{ TRANS}(R) b \Leftrightarrow b R a$$
- o produto das relações R e P - escrito como RP - é definido por:
$$a RP b \Leftrightarrow \exists c \in C \text{ tal que } a R c \text{ e } c P b$$

(notar que $R(PQ) = (RP)Q$, para quaisquer relações R, P e Q)

Gramáticas de precedência

Seja a gramática $G = (N, \Sigma, P, S)$. Sejam as relações:

- $X \text{ FIRST } Y \Leftrightarrow$ existe em P uma produção $X \rightarrow Y\alpha$, $\alpha \in (N \cup \Sigma)^*$
- $X \text{ LAST } Y \Leftrightarrow$ existe em P uma produção $X \rightarrow \alpha Y$, $\alpha \in (N \cup \Sigma)^*$

Vamos, então, construir as relações de precedência $<<, ==, >>$

1) a relação $==$ é trivial: basta examinar os lados direitos das produções da gramática.

2) para a relação $<<$ temos:

$$\begin{aligned} X << Y &\Leftrightarrow \exists A \rightarrow \alpha XB\beta \in P \text{ tal que } B \Rightarrow^+ Y\gamma \\ &\Leftrightarrow (X == B) \text{ e } (B \text{ FIRST}^+ Y) \\ &\Leftrightarrow X (==)(\text{FIRST}^+) Y \end{aligned}$$

Logo: $<< \equiv (==)(\text{FIRST}^+)$

3) para a relação $>>$ temos:

$$\begin{aligned} X >> a &\Leftrightarrow \exists A \rightarrow \alpha BY\beta \in P \text{ tal que } B \Rightarrow^+ \gamma X \text{ e } Y \Rightarrow^* a\delta \\ &\Leftrightarrow \exists B, Y \text{ tais que } (B \text{ LAST}^+ X), (B == Y), (Y \text{ FIRST}^* a) \\ &\Leftrightarrow \exists B \text{ tal que } (B \text{ LAST}^+ X), B (==)(\text{FIRST}^*) a \\ &\Leftrightarrow \exists B \text{ tal que } X \text{ TRANS}(\text{LAST}^+) B, B (==)(\text{FIRST}^*) a \\ &\Leftrightarrow X \text{ TRANS}(\text{LAST}^+)(==)(\text{FIRST}^*) a \end{aligned}$$

Logo: $>> \equiv \text{TRANS}(\text{LAST}^+)(==)(\text{FIRST}^*)$

Gramáticas de precedência

- ♦ Observação: Para construir a relação R^+ (fecho transitivo de R) a partir da relação R ver: GRIES - "Compiler construction for digital computers", Algoritmo de Warshall, p. 39. A relação R^* (fecho reflexivo e transitivo de R) é construída como $R^* = I + R^+$.
- ♦ Exemplo: Seja a gramática:
 $S \rightarrow a \mid (R)$
 $T \rightarrow S, T \mid S$
 $R \rightarrow T$

1) relação ==

	S	T	R	a	()	,
S							1
T							
R						1	
a							
(1				
)							
,		1					

Gramáticas de precedência

Gramática: $S \rightarrow a \mid (R)$
 $T \rightarrow S,T \mid S$
 $R \rightarrow T$

2) relação $<<$

a) FIRST

	S	T	R	a	()	,
S				1	1		
T	1						
R		1					
a							
(
)							
,							

b) FIRST⁺

	S	T	R	a	()	,
S				1	1		
T	1			1	1		
R	1	1		1	1		
a							
(
)							
,							

c) $(=)(FIRST^+)$

	S	T	R	a	()	,
S							
T							
R							
a							
(1	1		1	1		
)							
,	1			1	1		

Gramáticas de precedência

3) relação >>

a) LAST

	S	T	R	a	()	,
S				1		1	
T	1	1					
R		1					
a							
(
)							
,							

b) LAST⁺

	S	T	R	a	()	,
S				1		1	
T	1	1		1		1	
R	1	1		1		1	
a							
(
)							
,							

c) TRANS(LAST⁺)

	S	T	R	a	()	,
S		1	1				
T		1	1				
R							
a	1	1	1				
(
)	1	1	1				
,							

d) FIRST^{*}

	S	T	R	a	()	,
S	1			1	1		
T	1	1		1	1		
R	1	1	1	1	1		
a				1			
(1		
)						1	
,							1

e) TRANS(LAST⁺)(==)(FIRST^{*})

	S	T	R	a	()	,
S						1	
T						1	
R							
a						1	1
(
)						1	1
,							

Gramáticas de precedência

Portanto, para a gramática:

$$\begin{aligned} S &\rightarrow a \mid (R) \\ T &\rightarrow S, T \mid S \\ R &\rightarrow T \end{aligned}$$

a matriz de precedência será:

	S	T	R	a	()	,	\$
S						>>	==	>>
T						>>		>>
R						==		>>
a						>>	>>	>>
(<<	<<	==	<<	<<			>>
)						>>	>>	>>
,	<<	==		<<	<<			>>
\$	<<	<<	<<	<<	<<	<<	<<	

ALGORITMO:

1. inicializar;
2. testar se acabou (pilha = \$S e $i = |w| + 1$);
3. comparar pilha[topo] e $w[i]$:
 - a) se pilha[topo] << $w[i]$ ou pilha[topo] == $w[i]$ então empilhar;
 - b) se pilha[topo] >> $w[i]$ então reduzir;
 - c) caso contrário, erro.

SUBROTINA REDUZIR:

- ♦ procurar na pilha, a partir do topo, um par tal que pilha[j-1] << pilha[j]
- ♦ procurar produção $X \rightarrow \text{pilha[j]pilha[j+1]...pilha[topo]}$
- ♦ desempilhar pilha[j]pilha[j+1]...pilha[topo] e empilhar X.

Gramáticas de precedência

Exemplo: $w = (a, a)$

passo	entrada	pilha	observação
1	(a,a)\$	\$	
2	a,a)\$	(\$	
3	,a)\$	a(\$	
4	,a)\$	S(\$	reduzir: $S \rightarrow a$
5	a)\$,S(\$	
6)\$	a,S(\$	
7)\$	S,S(\$	reduzir: $S \rightarrow a$
8)\$	T,S(\$	reduzir: $T \rightarrow S$
9)\$	T(\$	reduzir: $T \rightarrow S,T$
10)\$	R(\$	reduzir: $R \rightarrow T$
11	\$)R(\$	
12	\$	S\$	reduzir: $S \rightarrow (R)$

Logo: sequência de derivações mais à direita:

$$S \Rightarrow (R) \Rightarrow (T) \Rightarrow (S,T) \Rightarrow (S,S) \Rightarrow (S,a) \Rightarrow (a,a)$$

Funções de precedência

- Uma matriz de precedência pode ocupar muito espaço. Em alguns casos é possível representar a informação da matriz de precedência por duas funções f e g tais que:

$X == Y$ implica em $f(X) = g(Y)$

$X << Y$ implica em $f(x) < g(Y)$

$X >> Y$ implica em $f(X) > g(Y)$

Quando isso for possível, iremos reduzir o espaço de armazenamento de n^2 para $2n$, onde n é o número de símbolos da gramática ($N \cup \Sigma$).

- Existem casos, contudo, para os quais não existem tais funções.

Exemplo:

	X	Y
X	==	>>
Y	==	==

$f(X) = g(X)$

$f(X) > g(Y)$

$f(Y) = g(X)$

$f(Y) = g(Y)$

$\Rightarrow f(X) > f(X)$ (contradição!)

ALGORITMO

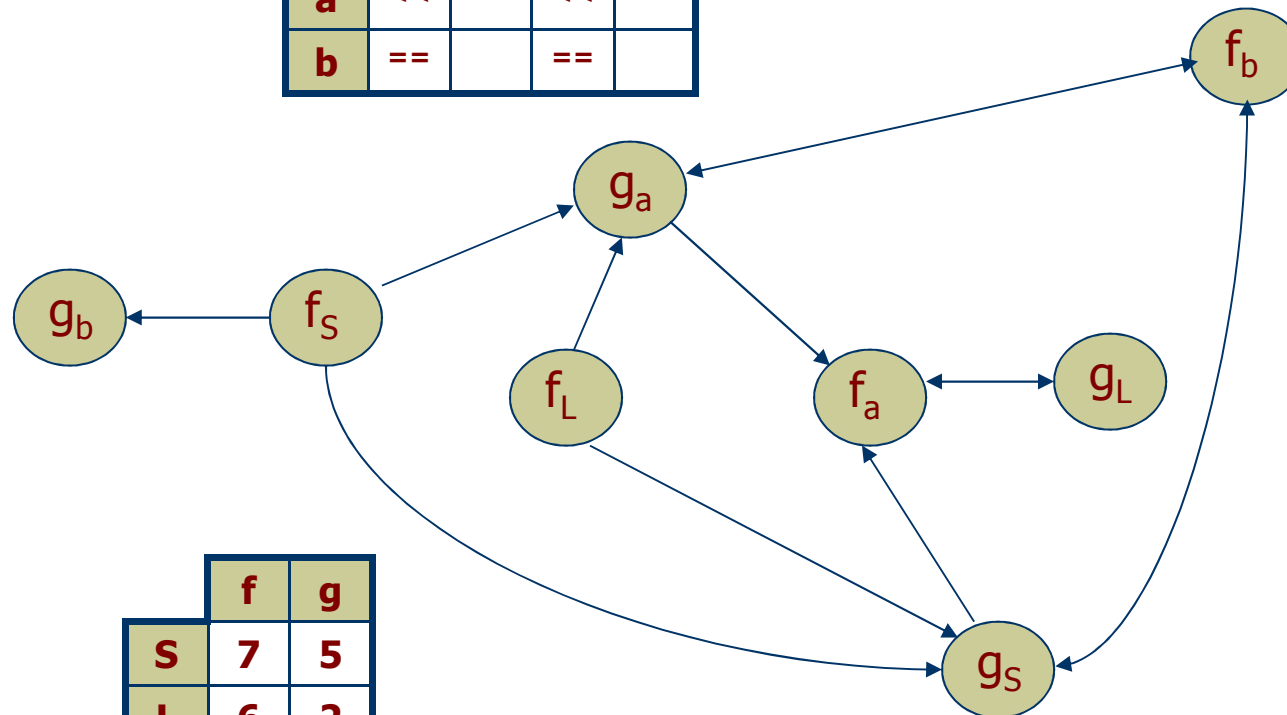
- Desenhar um grafo com $2n$ nós rotulados como $f_1, \dots, f_n, g_1, \dots, g_n$. Desenhar um arco orientado de f_i para g_j se $S_i >> S_j$ ou $S_i == S_j$. Desenhar um arco orientado de g_j para f_i se $S_i << S_j$ ou $S_i == S_j$.
- Testar por inconsistências. Caso não haja, atribuir a cada nó o número de nós distintos acessíveis a partir desse nó (incluindo o próprio nó).

Funções de precedência

- Exemplo: Seja uma gramática com a seguinte matriz de precedência:

	S	L	a	b
S	>>		>>	>>
L	>>		>>	
a	<<	==	<<	
b	==		==	

- Grafo:



- Logo:

	f	g
S	7	5
L	6	2
a	2	5
b	5	1