

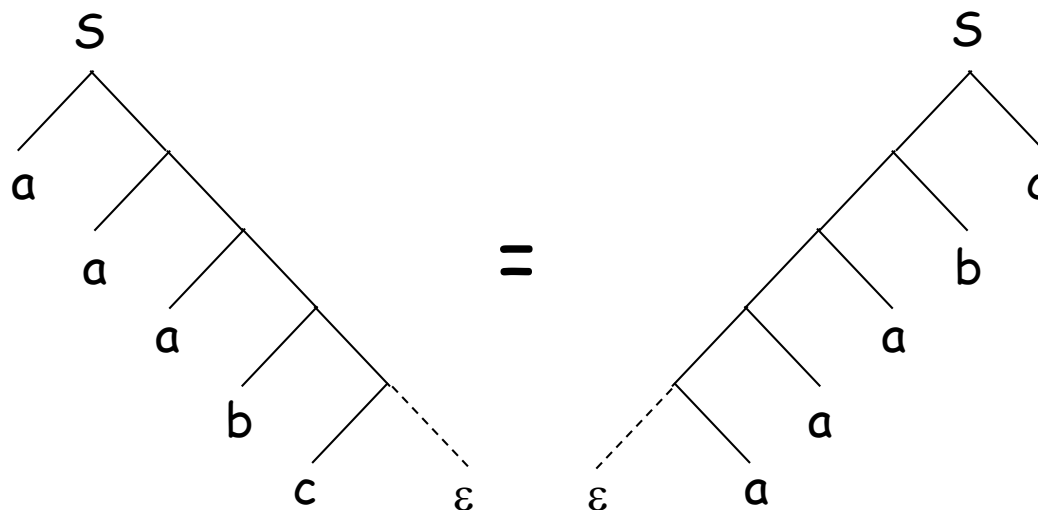
# Linguagens Formais e Autômatos (LFA)

Aula de 16/09/2013

**Conversões e Transformações**

## Conversões de Gramáticas LR em LL e vice-versa

Qual é a ideia?



Seja GRD uma gramática linear à direita: Existe uma gramática GRE linear à esquerda que gera o mesmo conjunto de cadeias que GRD?

## Uma exploração interessante

As regras de LR, gerada por GRD, têm o seguinte padrão:

a cadeia  $\beta$  de toda produção  $\alpha \rightarrow \beta$  é  $(\Sigma \cup \varepsilon) \cdot ((V - \Sigma) \cup \varepsilon)$

para  $\Sigma$  = alfabeto de LR e  $V$  = vocabulário da GRD que gera LR

Seja  $T$  a seguinte transformação definida sobre as produções de GRD:

Para toda produção  $\alpha \rightarrow \beta$  de GRD onde  $|\beta| > 1$

- criar uma produção  $\alpha \rightarrow \delta$  onde  $\delta$  é o inverso de  $\beta$

Para toda produção  $\alpha \rightarrow \beta$  de GRD onde  $|\beta| \leq 1$

- criar uma produção  $\alpha \rightarrow \delta$  idêntica a  $\alpha \rightarrow \beta$

Qual o perfil da linguagem  $L_T$  gerada pelas produções resultantes da transformações, se comparado a LR?

## Exemplo

LR

$S \rightarrow aS$

$S \rightarrow bX$

$X \rightarrow b$

$X \rightarrow c$

$L_\Gamma$

$S' \rightarrow S'a$

$S' \rightarrow X'b$

$X' \rightarrow b$

$X' \rightarrow c$

## Exemplo

LR

$S \rightarrow aS$

$S \rightarrow bX$

$X \rightarrow b$

$X \rightarrow c$

$L_T$

$S' \rightarrow S'a$

$S' \rightarrow X'b$

$X' \rightarrow b$

$X' \rightarrow c$

$a^*b(b|c)$

## Exemplo

LR

$$S \rightarrow aS$$

$$S \rightarrow bX$$

$$X \rightarrow b$$

$$X \rightarrow c$$

$$a^*b(b|c)$$

$L_T$

$$S' \rightarrow S'a$$

$$S' \rightarrow X'b$$

$$X' \rightarrow b$$

$$X' \rightarrow c$$

$$(b|c)ba^*$$

## Para gerar GRE a partir de GRD

Seja  $L$  uma linguagem LR e  $L^{\text{Rev}}$  e linguagem reversa de LR.

Seja  $\text{GRD}^{\text{Rev}}$  a gramática regular à direita capaz de gerar  $L^{\text{Rev}}$

A gramática regular à esquerda GRE, que gera as mesmas cadeias que LR, resulta das seguintes transformações sobre as produções de  $\text{GRD}^{\text{Rev}}$ :

Para toda produção  $\alpha \rightarrow \beta$  de  $\text{GRD}^{\text{Rev}}$  onde  $|\beta| > 1$

- criar uma produção  $\alpha \rightarrow \delta$  onde  $\delta$  é o inverso de  $\beta$

Para toda produção  $\alpha \rightarrow \beta$  de  $\text{GRD}^{\text{Rev}}$  onde  $|\beta| \leq 1$

- criar uma produção  $\alpha \rightarrow \delta$  idêntica a  $\alpha \rightarrow \beta$

Nosso desafio:

Encontrar  $\text{GRD}^{\text{Rev}}$ , a gramática regular à direita que gera  $L^{\text{Rev}}$

## O valor de AF na busca da versão LL de LR

1. Para encontrar a linguagem reversa de LR, invertemos a GRD que gera LR (= GRE de  $L^{\text{Rev}}$ )

GRD que gera LR

$$\begin{array}{l} S \rightarrow aS \\ S \rightarrow bX \\ X \rightarrow b \\ X \rightarrow c \end{array} \}$$

Note-se que as expansões do símbolo RAIZ ("S") numa LR, quando incluem um terminal, indicam o PREFIXO das cadeias da linguagem que definem.

GRE que gera  $L^{\text{Rev}}$

$$\begin{array}{l} S' \rightarrow S'a \\ S' \rightarrow X'b \\ X' \rightarrow b \\ X' \rightarrow c \end{array} \}$$

As expansões do símbolo RAIZ ("S") numa LL, quando incluem um terminal, indicam o SUFIXO das cadeias da linguagem que definem.



## O valor de AF na busca da versão LL de LR

2. Para encontrar  $GRD^{Rev}$  (gramática regular à direita que gera  $L^{Rev}$ ) podemos usar o AF de GRE:

GRE

$S' \rightarrow S'a$

$S' \rightarrow X'b$

$X' \rightarrow b$

$X' \rightarrow c$

$(b|c)ba^*$

$AF_{Rev}$

estado inicial :  $q_0$

estado final :  $q_2$

Transições:

$q_0, b, q_1 \dots S \rightarrow b X$

$q_0, c, q_1 \dots S \rightarrow c X$

$q_1, b, q_2 \dots X \rightarrow b Y$

$q_2, a, q_2 \dots Y \rightarrow a Y \mid \epsilon$

$(b|c)ba^*$



## O valor de AF na busca da versão LL de LR

3. A GRE que gera o mesmo conjunto de cadeias de LR é o resultado da inversão das regras de  $GRD^{Rev}$

$GRD^{Rev}$

$S \rightarrow b X$

$S \rightarrow c X$

$X \rightarrow b Y$

$Y \rightarrow a Y$

$Y \rightarrow \varepsilon$

$(b|c)ba^*$

$GRE_{Final}$

$S \rightarrow X b$

$S \rightarrow X c$

$X \rightarrow Y b$

$Y \rightarrow Y a$

$Y \rightarrow \varepsilon$

$a^*b(b|c)$

## Aspectos interessantes de estruturas LR e LL

A linguagem de programação PROLOG (baseada em lógica, muito utilizada em aplicações de Inteligência Artificial), tem facilidades interessantes para se trabalhar com 'Gramáticas'.

Uma destas facilidades é um formalismo denominado "Definite Clause Grammars" ou DCG.

Neste formalismo, a GRD inicial (que gera LR) se escreve assim:

$s \rightarrow [a], s.$

$s \rightarrow [b], x.$

$x \rightarrow [b].$

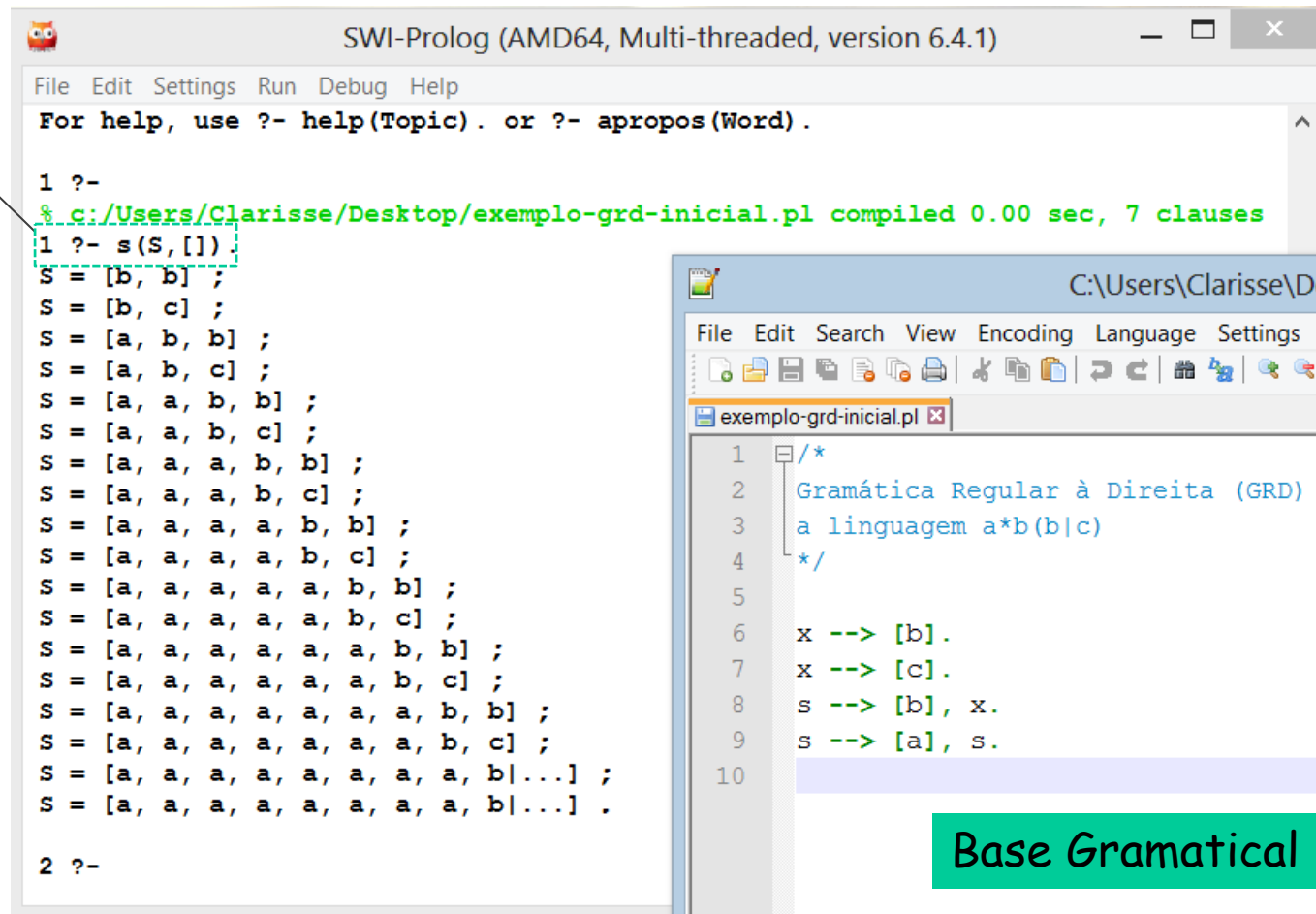
$x \rightarrow [c].$

O uso de '[' ]' marca os *terminais* do alfabeto de GRD. Os demais literais marcam os não terminais do vocabulário de GRD.

# Execução de uma "consulta à base gramatical"

$s(S, [])$ .

Mostre TODAS as formas de instanciar a variável "S" de tal modo a SATISFAZER a(s) regra(s) relativa(s) ao nó "s" da gramática.



The image shows two windows. The top window is SWI-Prolog (AMD64, Multi-threaded, version 6.4.1) with the following content:

```

File Edit Settings Run Debug Help
For help, use ?- help(Topic) . or ?- apropos(Word) .

1 ?-
% c:/Users/Clarisse/Desktop/exemplo-grd-inicial.pl compiled 0.00 sec, 7 clauses
1 ?- s(S, []).
S = [b, b] ;
S = [b, c] ;
S = [a, b, b] ;
S = [a, b, c] ;
S = [a, a, b, b] ;
S = [a, a, b, c] ;
S = [a, a, a, b, b] ;
S = [a, a, a, b, c] ;
S = [a, a, a, a, b, b] ;
S = [a, a, a, a, b, c] ;
S = [a, a, a, a, a, b, b] ;
S = [a, a, a, a, a, b, c] ;
S = [a, a, a, a, a, a, b, b] ;
S = [a, a, a, a, a, a, b, c] ;
S = [a, a, a, a, a, a, a, b, b] ;
S = [a, a, a, a, a, a, a, b, c] ;
S = [a, a, a, a, a, a, a, a, b, ...] ;
S = [a, a, a, a, a, a, a, a, a, b, ...] .

2 ?-
  
```

The bottom window is a text editor showing the grammar definition:

```

C:\Users\Clarisse\De
File Edit Search View Encoding Language Settings
exemplo-grd-inicial.pl x
1 /*
2 Gramática Regular à Direita (GRD)
3 a linguagem a*b(b|c)
4 */
5
6 x --> [b].
7 x --> [c].
8 s --> [b], x.
9 s --> [a], s.
10
  
```

A green box labeled "Base Gramatical" is positioned at the bottom right of the text editor window.

## Como o Prolog busca “respostas” para consultas

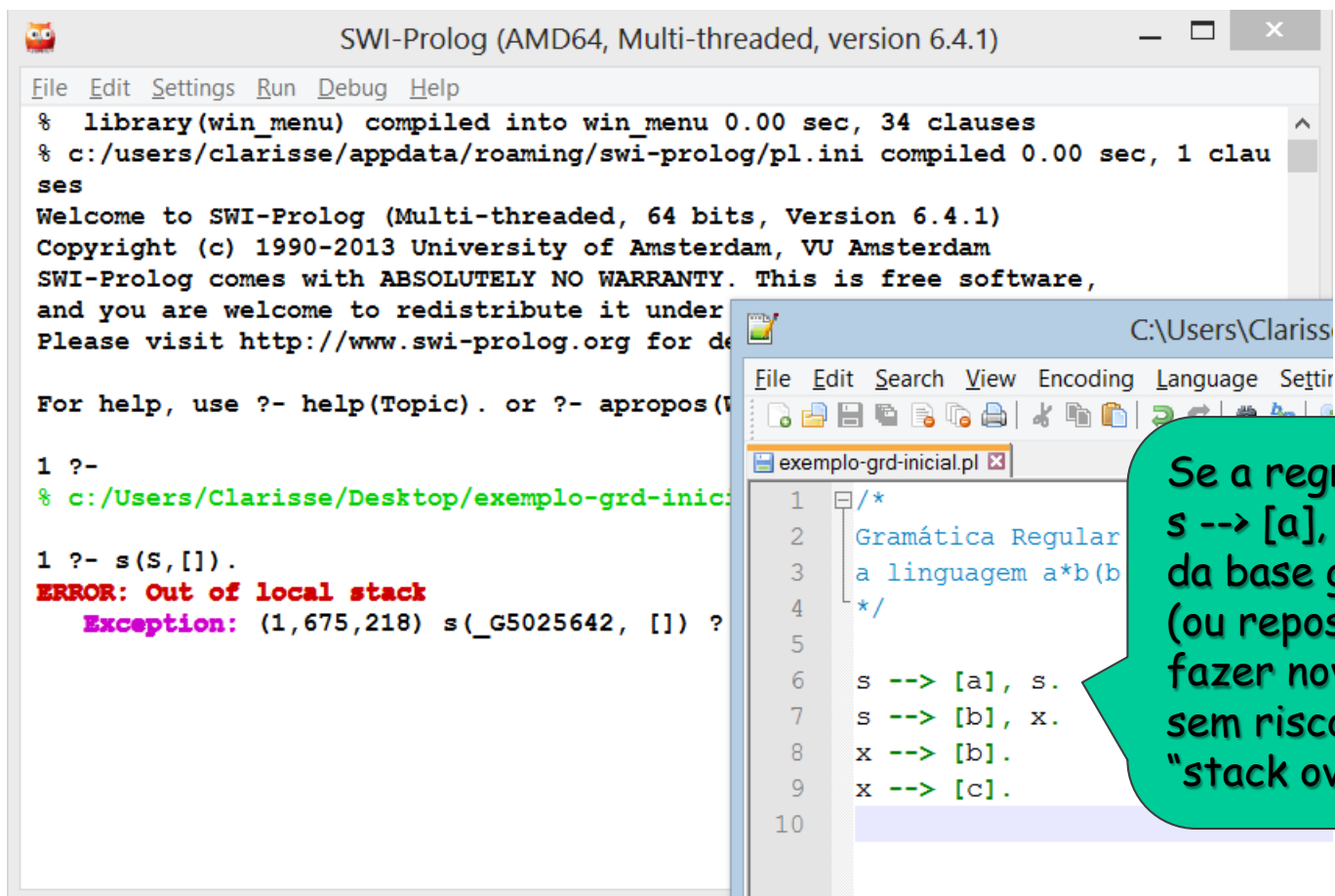
Uma consulta constitui uma “meta” a ser provada.

O Prolog busca satisfazer a “meta” em questão, varrendo toda a sua base de fatos e regras, de cima para baixo, da esquerda para a direita.

Ao encontrar uma condição que satisfaz a meta, uma resposta é produzida e a busca por uma nova resposta possível é retomada.

Observem no próximo slide o que ocorre se alteramos a ordem das cláusulas na base gramatical do programa.

## “Stack Overflow” se 1ª cláusula da base é recursiva



The screenshot shows the SWI-Prolog (AMD64, Multi-threaded, version 6.4.1) window. The main window displays the Prolog prompt and the user's input. The user has entered the query `1 ?- s(S, []).`, which has resulted in a stack overflow error. The error message is `ERROR: Out of local stack` and `Exception: (1,675,218) s(_G5025642, []) ?`. The file editor window shows the Prolog code for the grammar rules.

```
File Edit Settings Run Debug Help
% library(win_menu) compiled into win_menu 0.00 sec, 34 clauses
% c:/users/clarisse/appdata/roaming/swi-prolog/pl.ini compiled 0.00 sec, 1 clause
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.4.1)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Topic).

1 ?-
% c:/Users/Clarisse/Desktop/exemplo-grd-inicial.pl

1 ?- s(S, []).
ERROR: Out of local stack
Exception: (1,675,218) s(_G5025642, []) ?
```

```
File Edit Search View Encoding Language Settings
exemplo-grd-inicial.pl
1 /*
2 Gramática Regular
3 a linguagem a*b(b
4 */
5
6 s --> [a], s.
7 s --> [b], x.
8 x --> [b].
9 x --> [c].
10
```

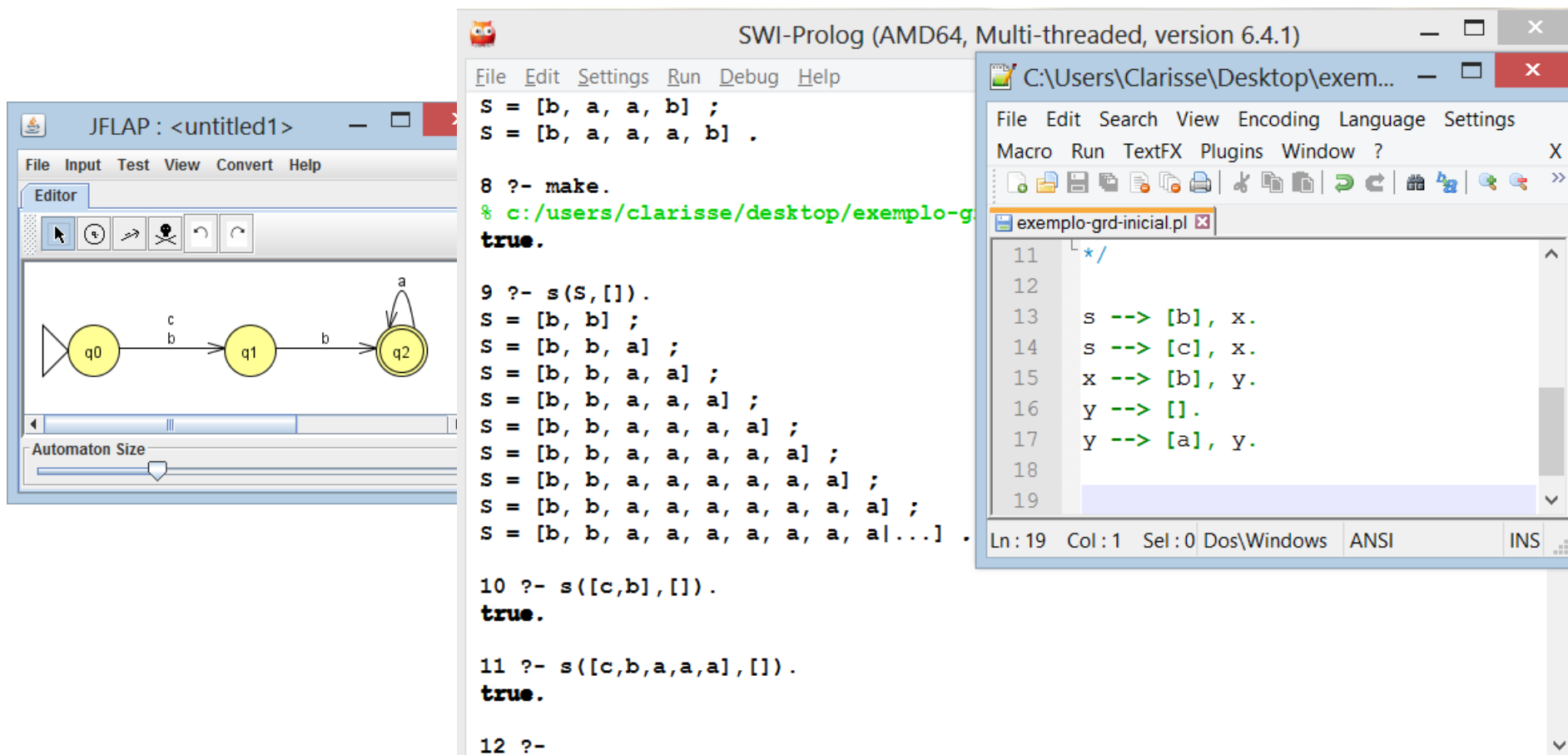
Se a regra recursiva  $s \rightarrow [a], s.$  for retirada da base gramatical (ou reposicionada) podem-se fazer novamente consultas sem riscos de “stack overflow”.

## A GRE gera a linguagem reversa de LR ( $L^{Rev}$ )

The screenshot shows the SWI-Prolog IDE window titled "SWI-Prolog (AMD64, Multi-threaded, version 6.4.1)". The menu bar includes File, Edit, Settings, Run, Debug, and Help. The main editor area displays the following text:

```
4 ?- make.  
% c:/users/clarisse/desktop/exemplo-grd-inicial compiled 0.00 sec, 3 clauses  
true.  
  
5 ?- s(S,[ ]).  
S = [c, b] ;  
S = [b, b] ;  
S = [c, b, a] ;  
S = [b, b, a] ;  
S = [c, b, a, a] ;  
S = [b, b, a, a] ;  
S = [c, b, a, a, a] ;  
S = [b, b, a, a, a] ;  
S = [c, b, a, a, a, a] ;  
S = [b, b, a, a, a, a] ;  
S = [c, b, a, a, a, a, a] ;  
S = [b, b, a, a, a, a, a] ;  
S = [c, b, a, a, a, a, a, a] ;  
S = [b, b, a, a, a, a, a, a] ;  
S = [c, b, a, a, a, a, a, a, a] ;  
S = [b, b, a, a, a, a, a, a, a] ;  
S = [c, b, a, a, a, a, a, a, a, a] ;  
S = [b, b, a, a, a, a, a, a, a, a] ;  
S = [c, b, a, a, a, a, a, a, a, a, a] ;  
S = [b, b, a, a, a, a, a, a, a, a, a] ;  
S = [c, b, a, a, a, a, a, a, a, a, a, a] .
```

A  $GRD^{Rev}$  (GRD que gera a linguagem  $L^{Rev}$ )





A GRE equivalente à GRD, ambas gerando  $a^*b(b|c)$

SWI-Prolog (AMD64, Multi-threaded, version 6.4.1)

```
File Edit Settings Run Debug Help
```

```
S = [b, b, a] ;
S = [b, b, a, a] ;
S = [b, b, a, a, a] .

16 ?- make.
% c:/users/clarisse/desktop/exemplo-grd-inicial.pl
true.

17 ?- s(S, []).
S = [b, b] ;
S = [a, b, b] ;
S = [a, a, b, b] ;
S = [a, a, a, b, b] ;
S = [a, a, a, a, b, b] ;
S = [a, a, a, a, a, b, b] ;
S = [a, a, a, a, a, a, b, b] ;
S = [a, a, a, a, a, a, a, b, b] ;
S = [a, a, a, a, a, a, a, a, b, b] ;
S = [a, a, a, a, a, a, a, a, a, b, b] .

18 ?- s([b,c], []).
ERROR: Out of local stack
Exception: (6) s([b, c], []) ? abort
% Execution Aborted
19 ?- s([a,b,b], []).
true
```

C:\Users\Clarisse\Desktop\exem... File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?

exemplo-grd-inicial.pl

```
11 */
12
13 s --> x, [b]. %s --> [b], x.
14 s --> x, [c]. %s --> [c], x.
15 x --> y, [b]. %x --> [b], y.
16 y --> [].
17 y --> y, [a]. %y --> [a], y.
18
```

Ln: 17 Col: 14 Sel: Dos\Windows ANSI INS

Quase... Por que o "stack overflow"?

Várias ordens alternativas das regras vão sempre acabar em "overflow". A menos de um desvio de controle na estrutura de execução, o Prolog tem problemas com estruturas que se expandem "à esquerda" (ou "primeiro" [antes] do que uma solução que é terminal, não se expande). O teste de alternativas frequentemente leva a "overflow".

Eis um exemplo de por que interessa saber e poder converter LL's em LR's!  
Se recebemos uma base LL para trabalhar, podemos transformá-la em LR.  
Da mesma forma perceber que as condições de parada devem preceder as de expansão é fundamental.

## Exercícios

Criar uma GRE para LR's definidas a seguir:

1.  $S \rightarrow a A; A \rightarrow b C; A \rightarrow b D; C \rightarrow c C; C \rightarrow c; D \rightarrow d$

2.  $(a(b|c)^*)^+ | d^*e^+$

3. AF

