CM 4 Compilation : bottom-up Parsers (LR)

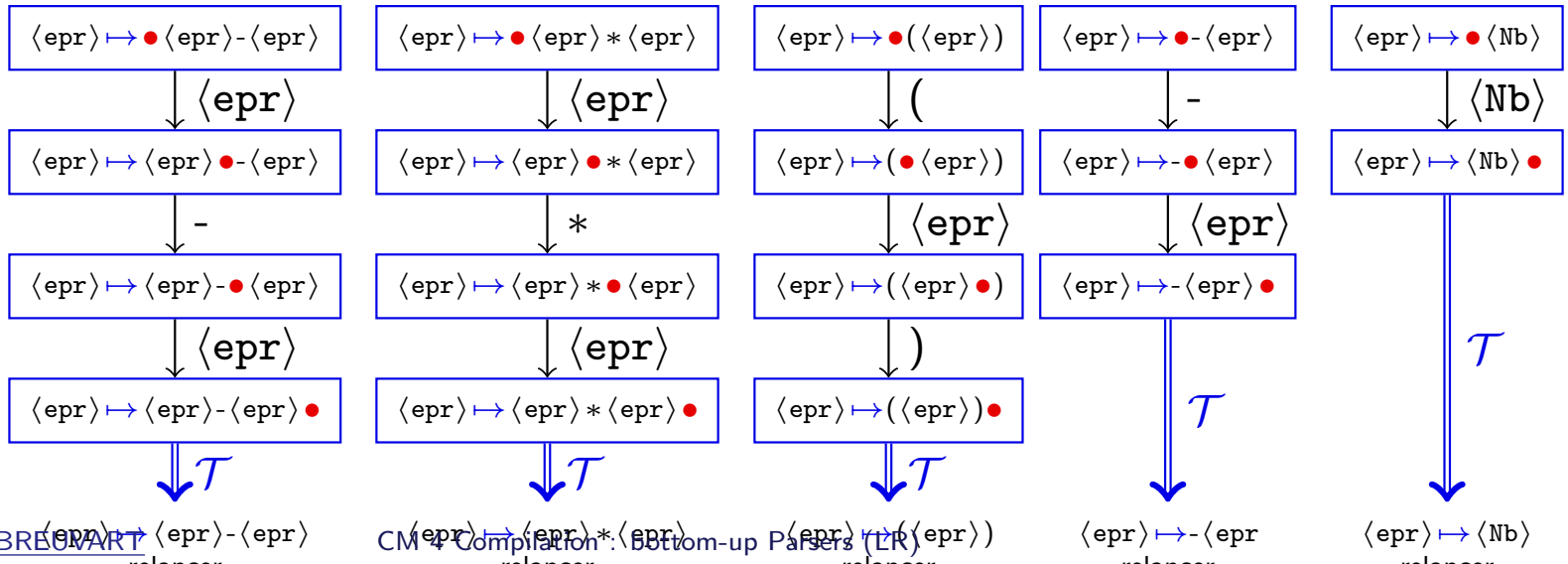
Flavien BREUVART

February 12, 2026

The Non-Deterministic Automaton

How to create the non-determnistic LR (NDLR) ?

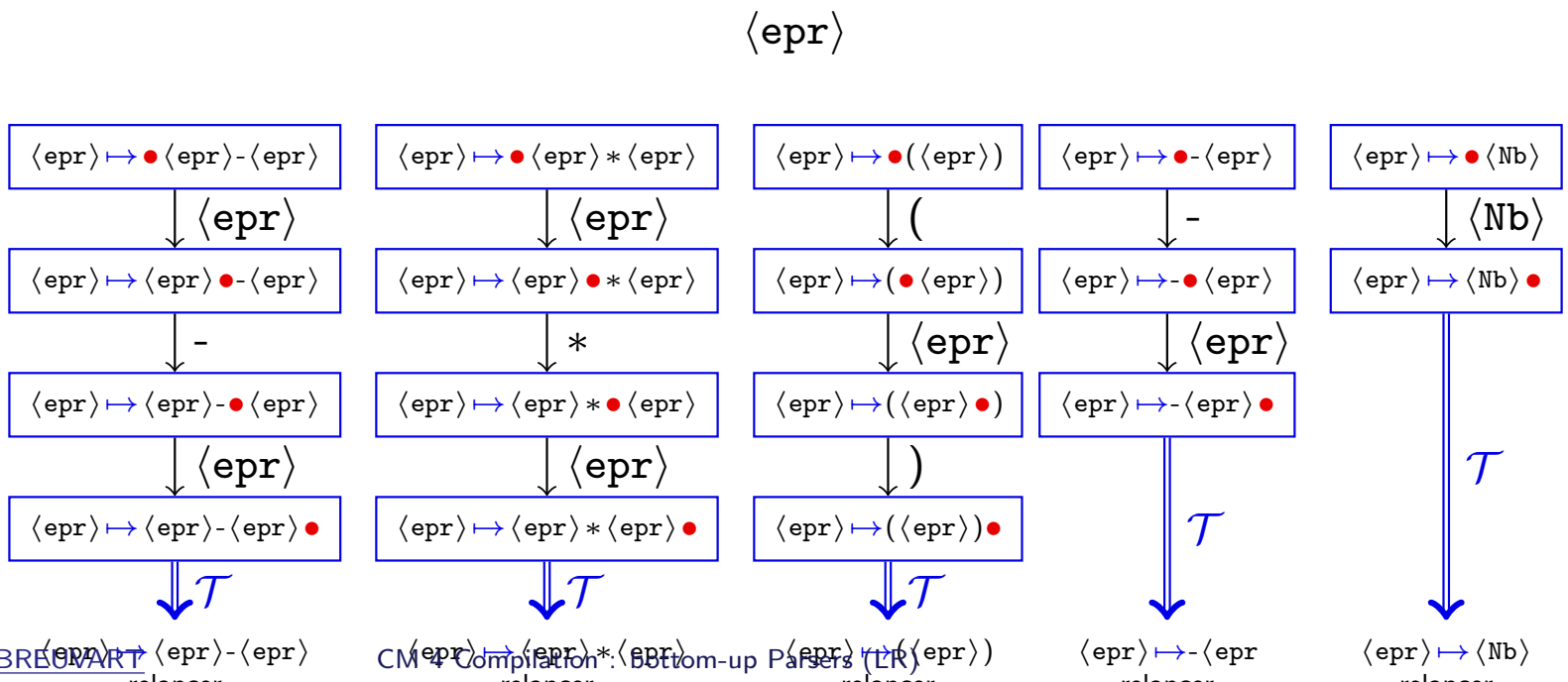
- shifts reading each rules,



The Non-Deterministic Automaton

How to create the non-determnistic LR (NDLR) ?

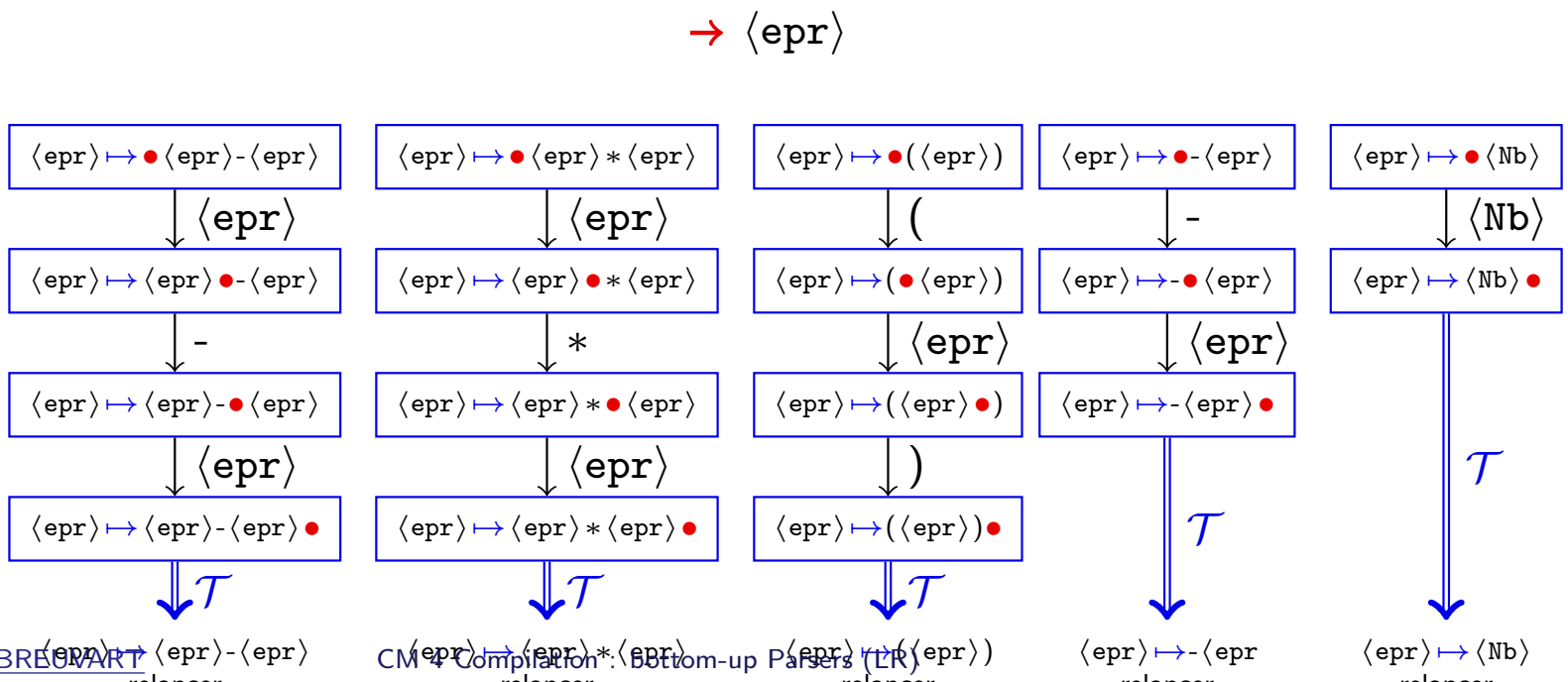
- shifts reading each rules,
- an additional state for each non-terminal N,



The Non-Deterministic Automaton

How to create the non-determnistic LR (NDLR) ?

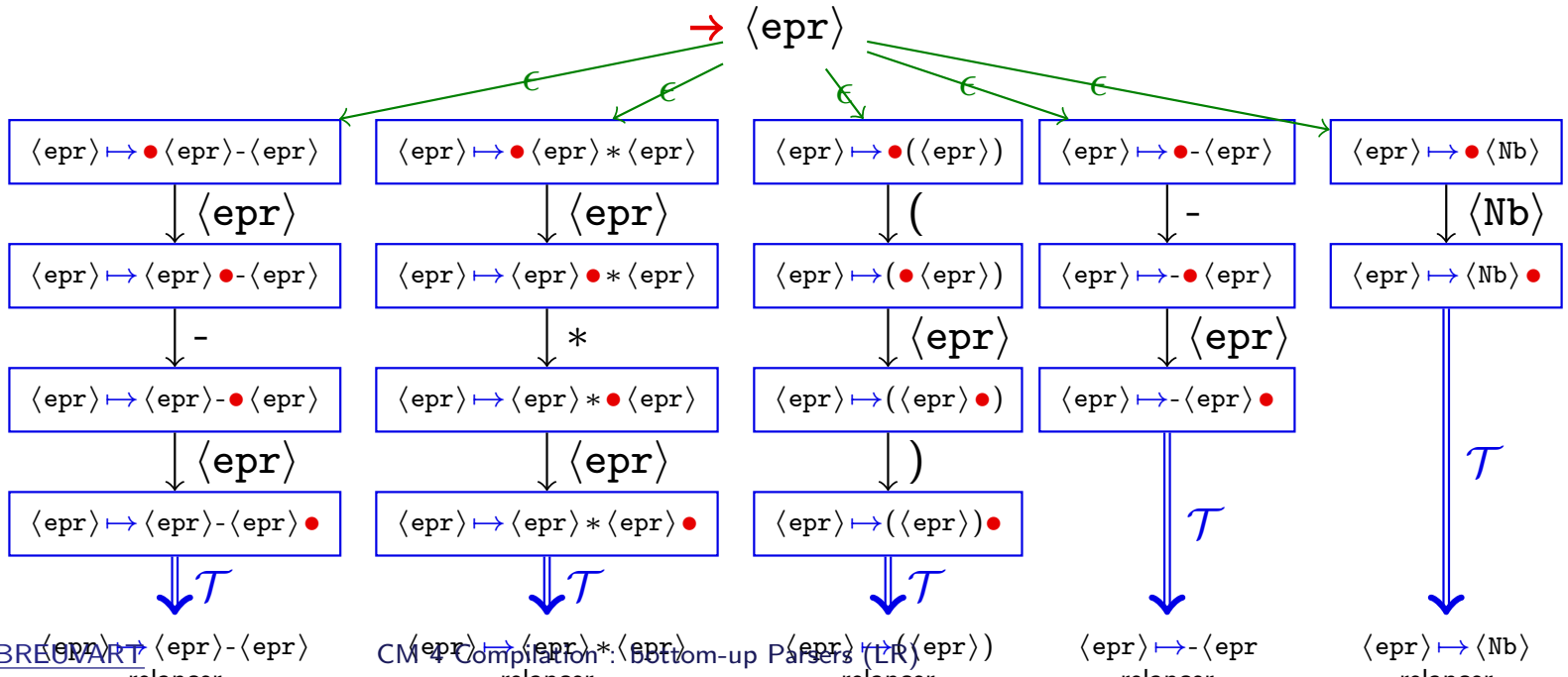
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,



The Non-Deterministic Automaton

How to create the non-determnistic LR (NDLR) ?

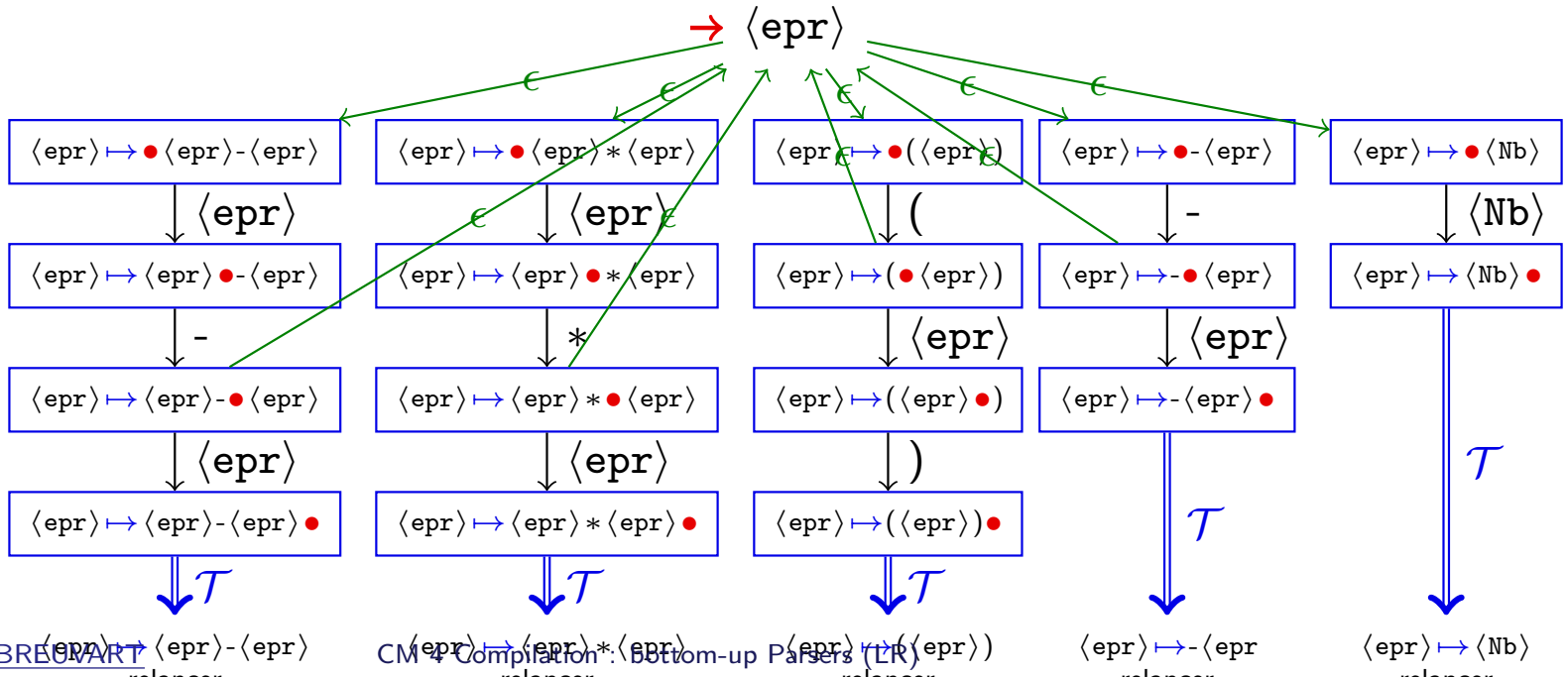
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,
- add ϵ-transitions toward each rule,



The Non-Deterministic Automaton

How to create the non-determnistic LR (NDLR) ?

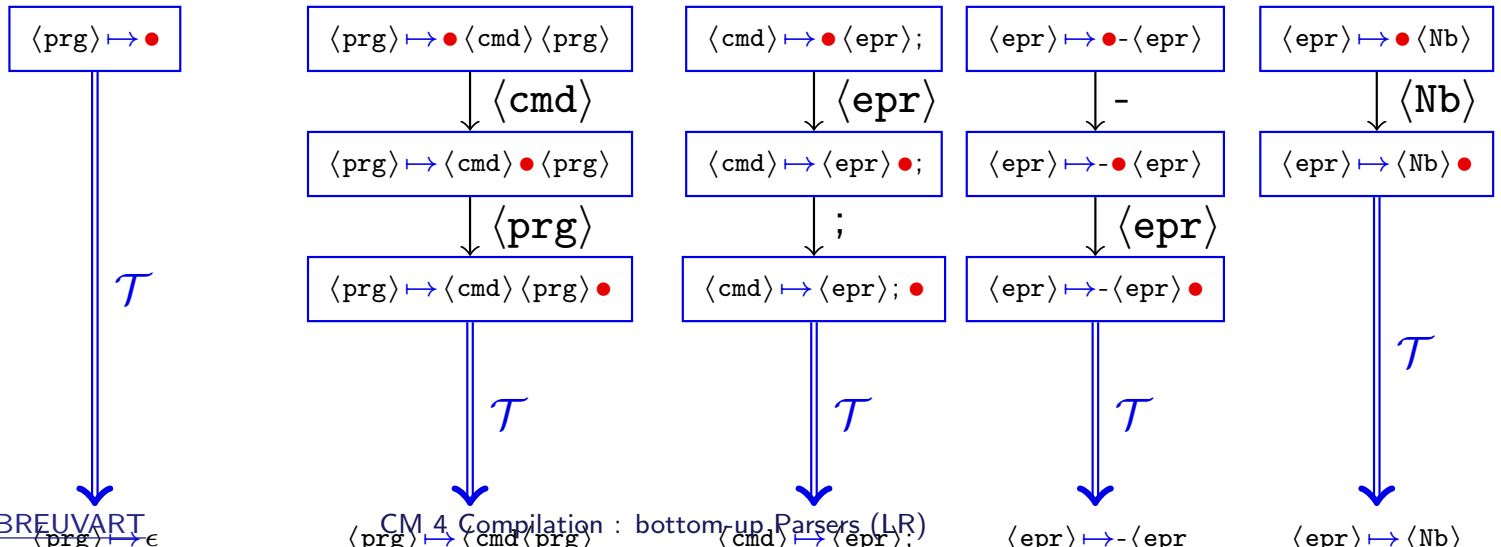
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,
- add ϵ-transitions toward each rule,
- for each state $M \mapsto w_1 \bullet N w_2$, add an ϵ-transition toward N



The Non-Deterministic Automaton, again

How to create the non-determnistic LR (NDLR) ?

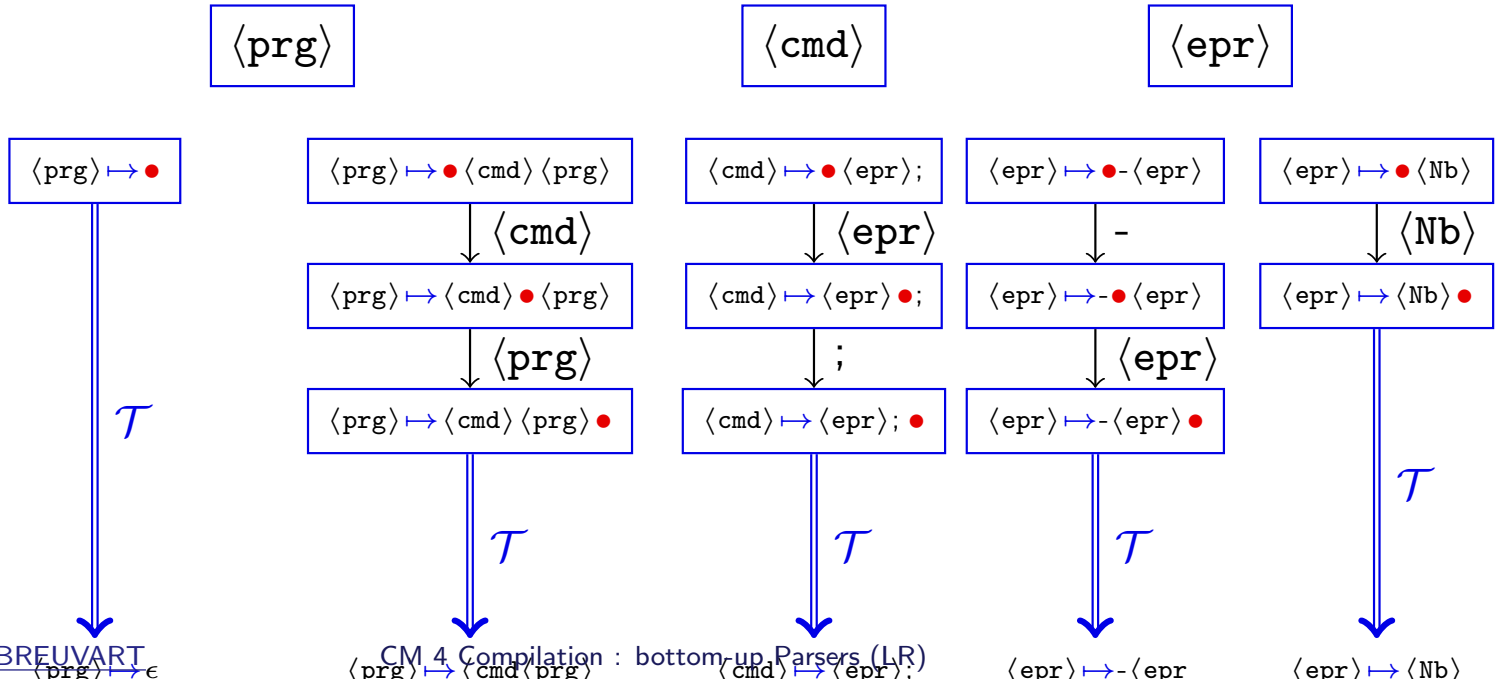
- shifts reading each rules,



The Non-Deterministic Automaton, again

How to create the non-determnistic LR (NDLR) ?

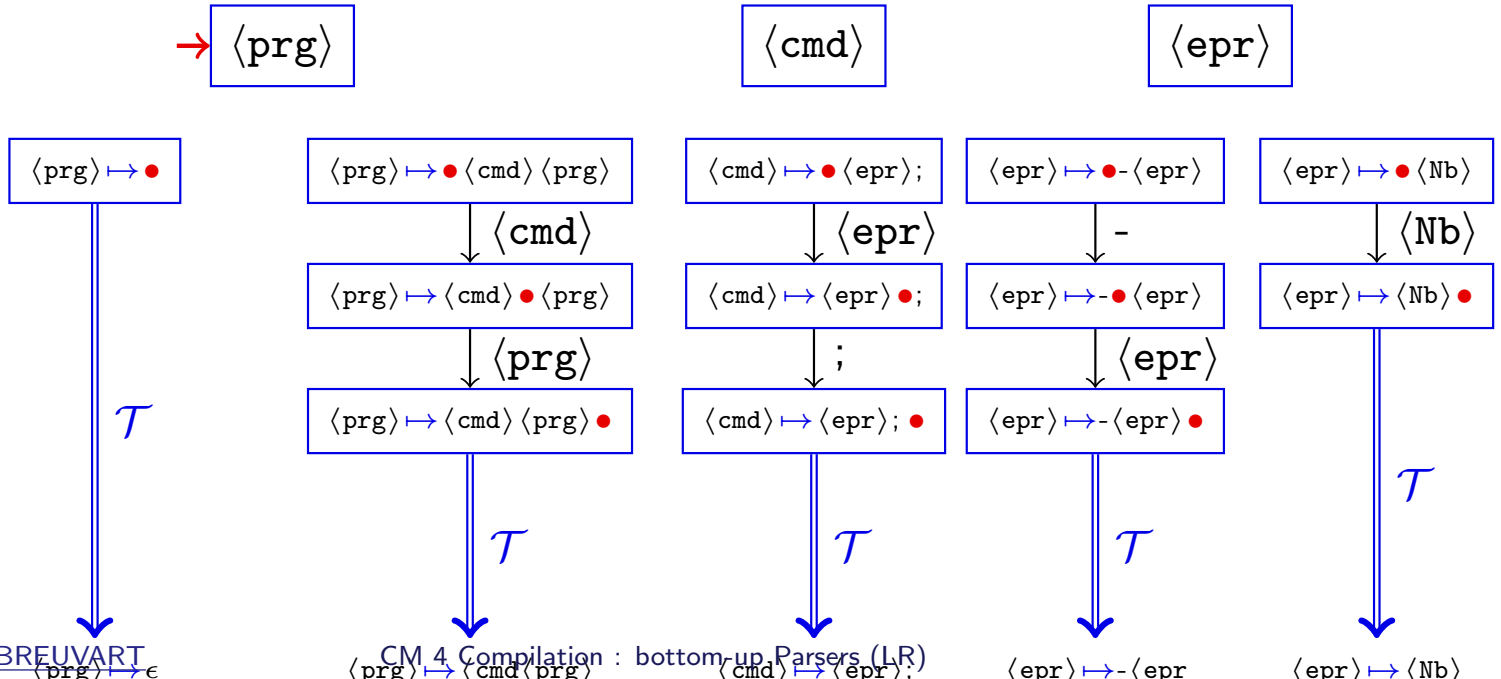
- shifts reading each rules,
- an additional state for each non-terminal N,



The Non-Deterministic Automaton, again

How to create the non-determnistic LR (NDLR) ?

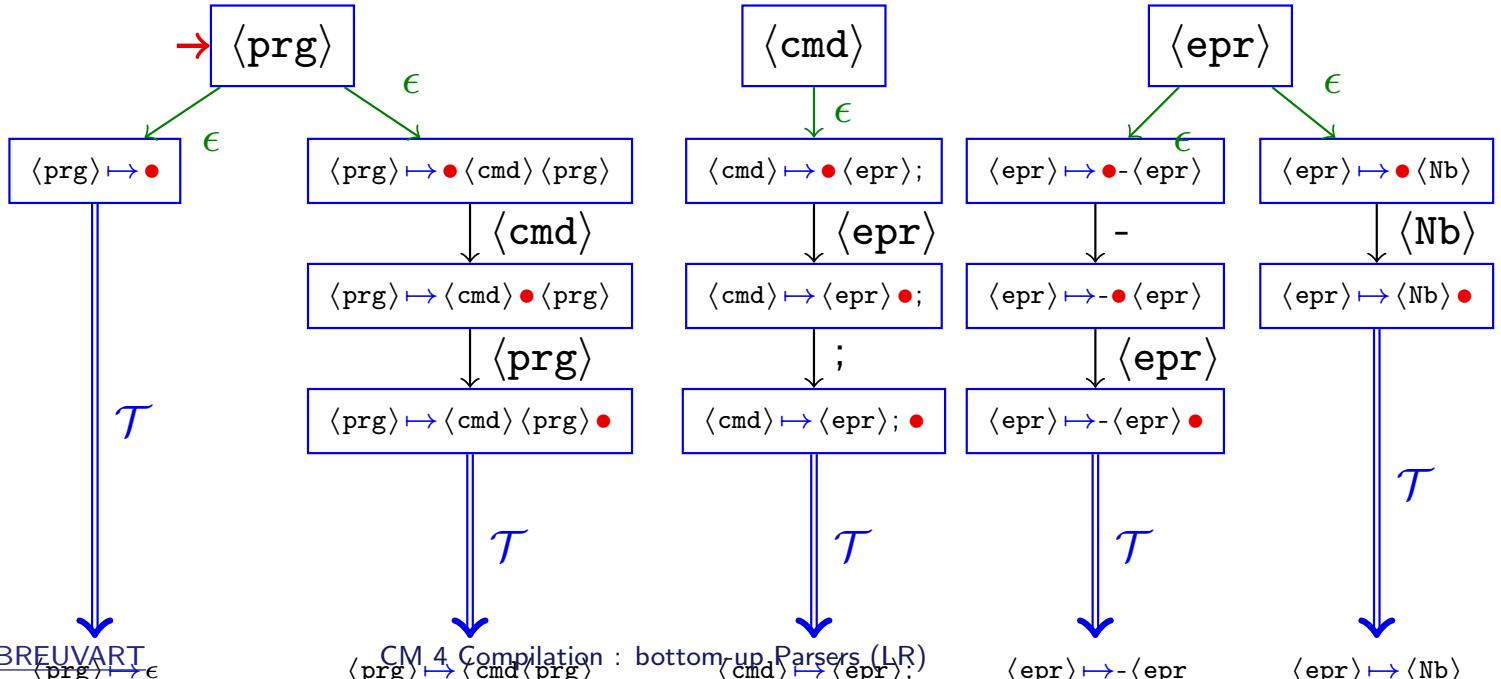
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,



The Non-Deterministic Automaton, again

How to create the non-determnistic LR (NDLR) ?

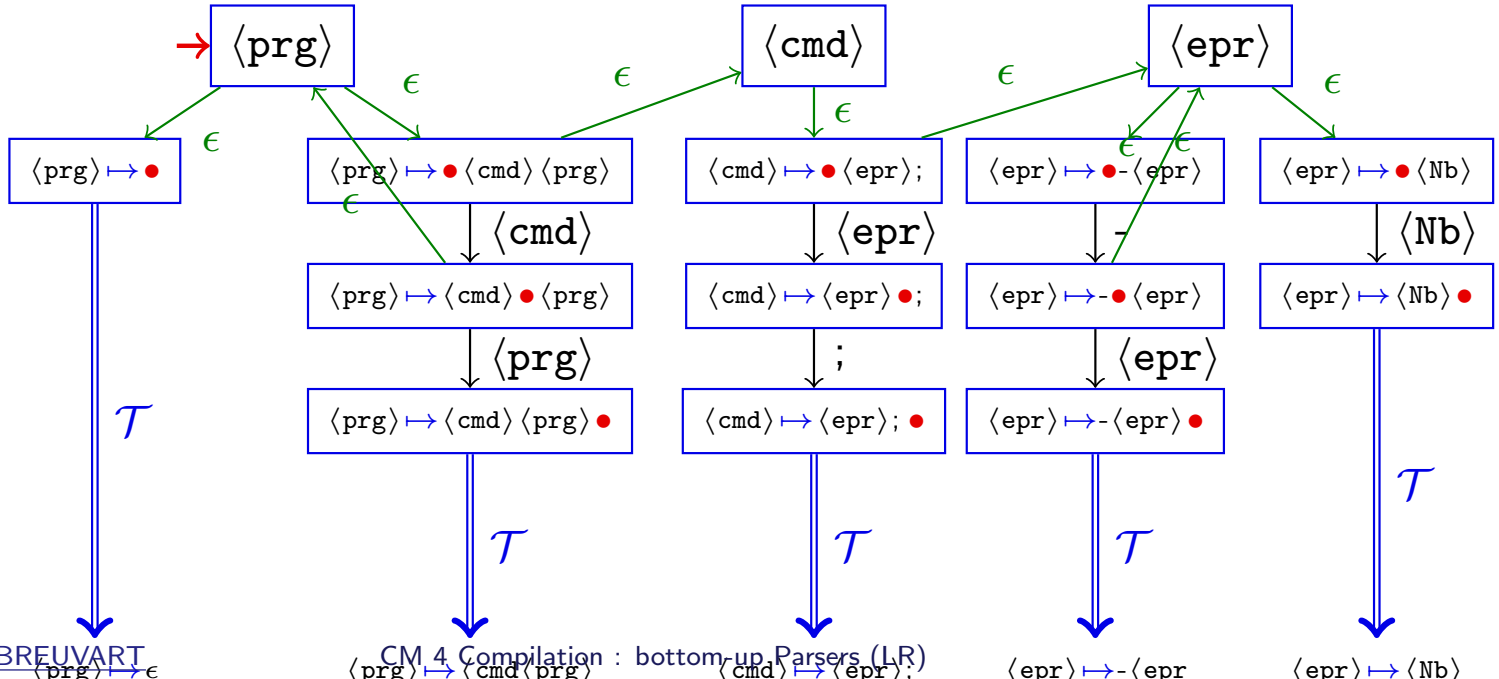
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,
- add ϵ-transitions toward each rule,



The Non-Deterministic Automaton, again

How to create the non-determnistic LR (NDLR) ?

- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,
- add ϵ-transitions toward each rule,
- for each state $M \mapsto w_1 \bullet N w_2$, add an ϵ-transition toward N

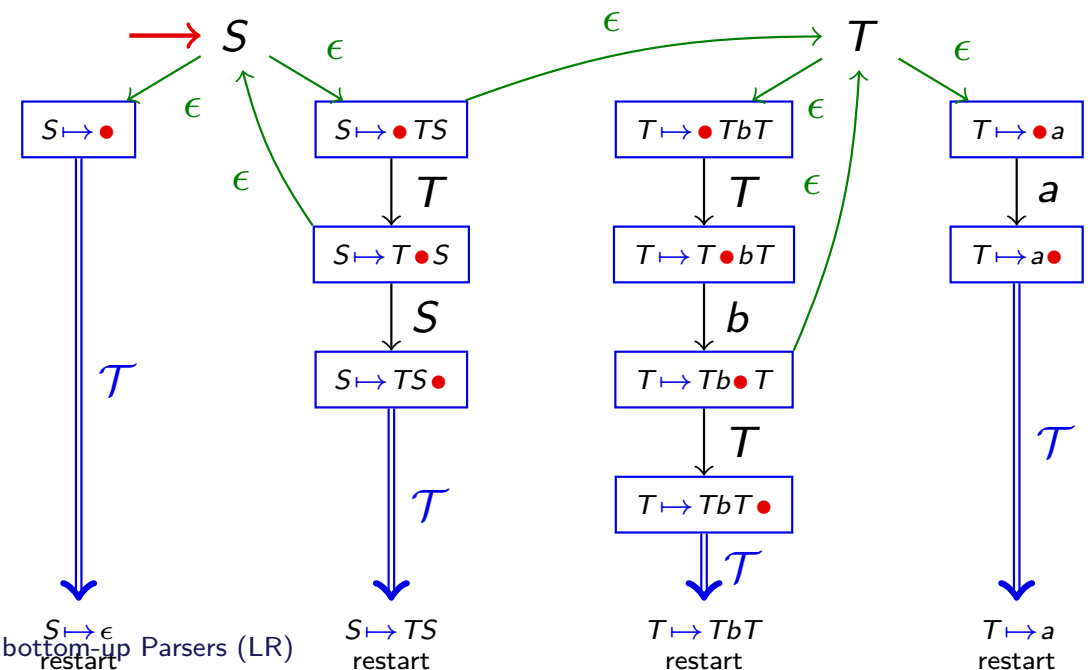


The Non-Deterministic Automaton, again

And again...

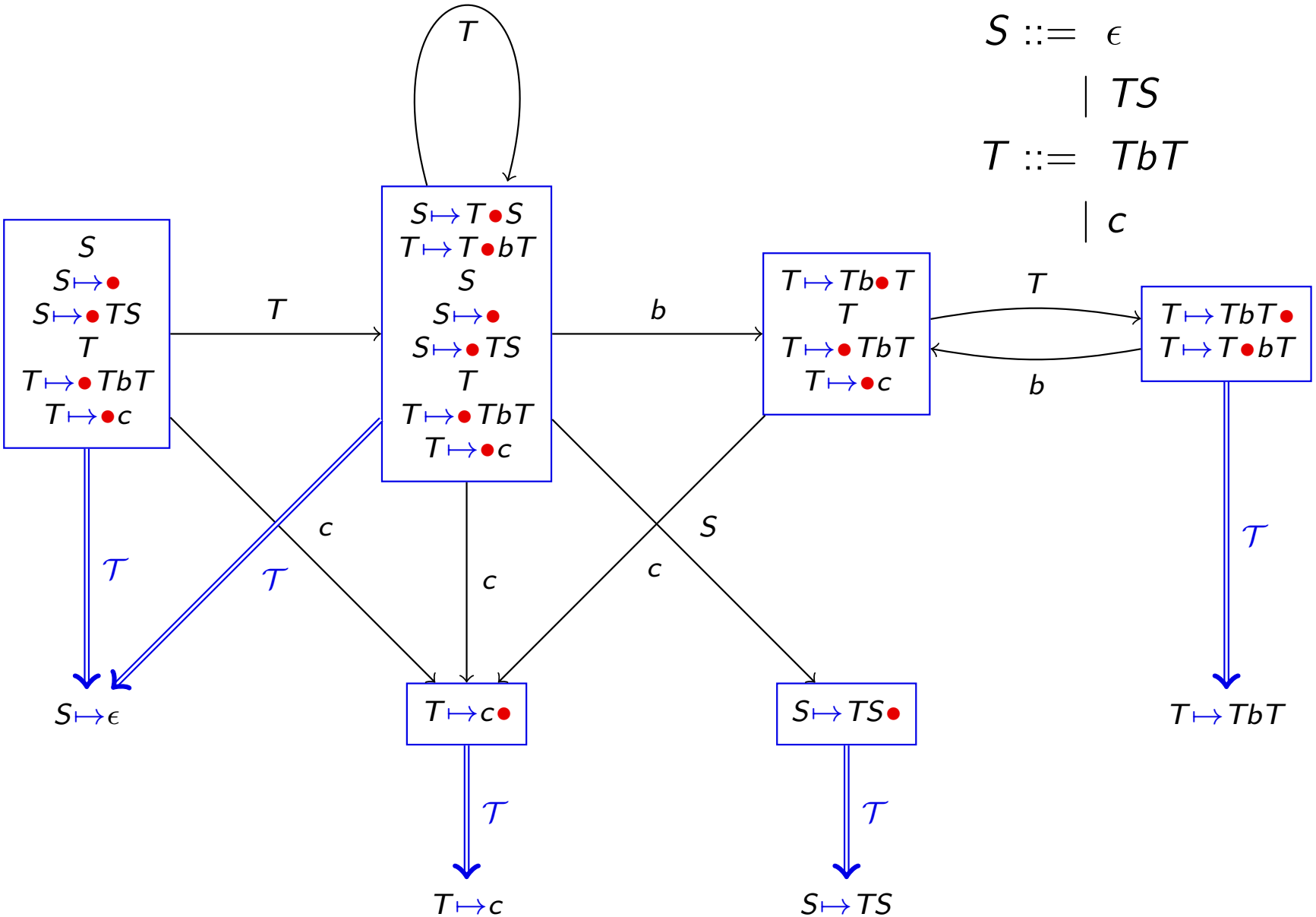
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,
- add ϵ-transitions toward each rule,
- for each state $M \mapsto w_1 \bullet N w_2$, add an ϵ-transition toward N

$$
\begin{aligned}
S ::= &\ \epsilon \\
&\mid TS \\
T ::= &\ TbT \\
&\mid a
\end{aligned}
$$



Pseudo-determinisation : Parser LR_0

$$S ::= \epsilon \mid TS$$
$$T ::= TbT \mid c$$



In order to gain time during exam: Perform the determinisation on-the-fly

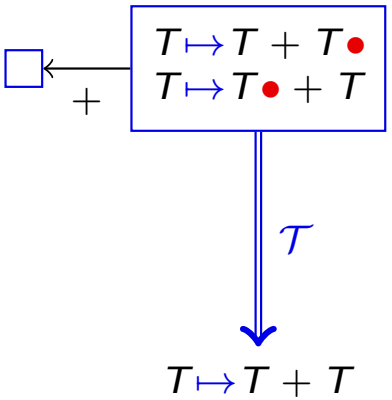
Explained during TD.

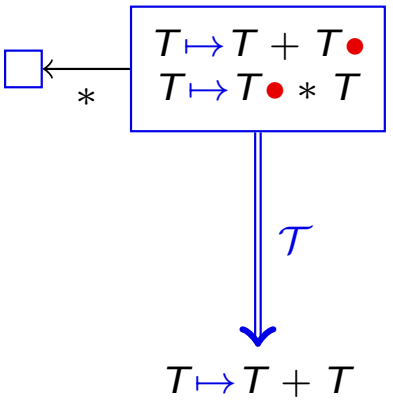
Idea

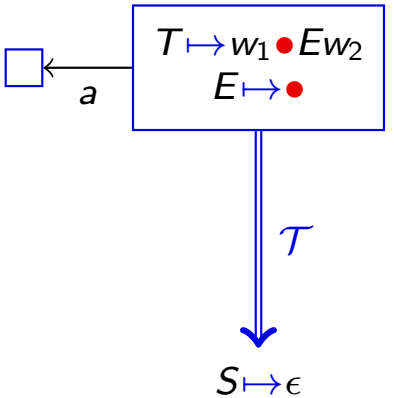
Using default names for the non-deterministic version

S
$S \mapsto \bullet$
$S \mapsto \bullet TS$
T
$T \mapsto \bullet TbT$
$T \mapsto \bullet c$

Some usual conflicts







Associativity

right associativity :
↳ choose the shift

left associativity :
↳ choose the action

Priority

$*$ priority :
↳ choose the shift

$+$ priority :
↳ choose the action

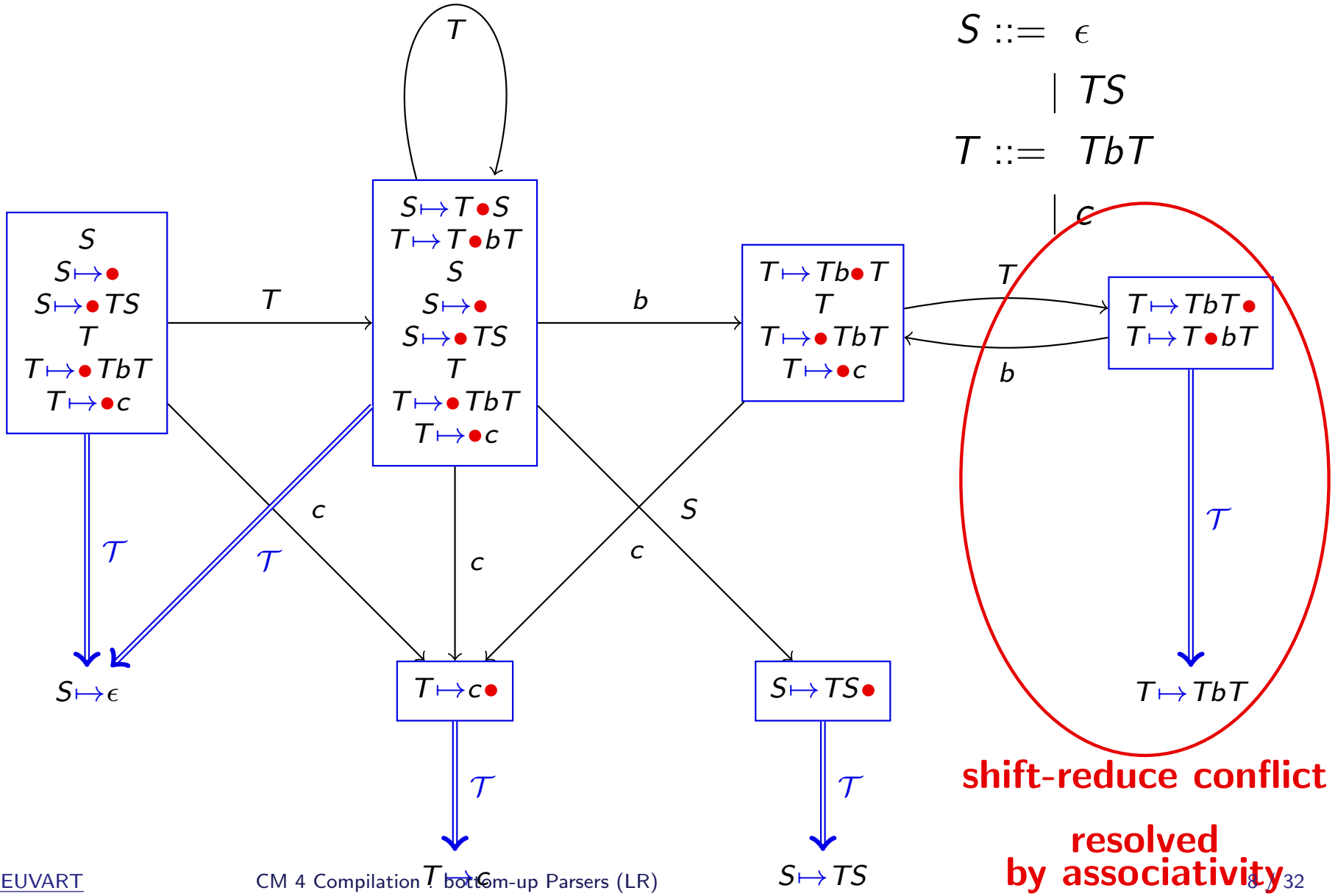
Artificial conflicts

No ambiguity (No c or T can follow aS)

Need of stronger algo
↳ often resolved using SLR

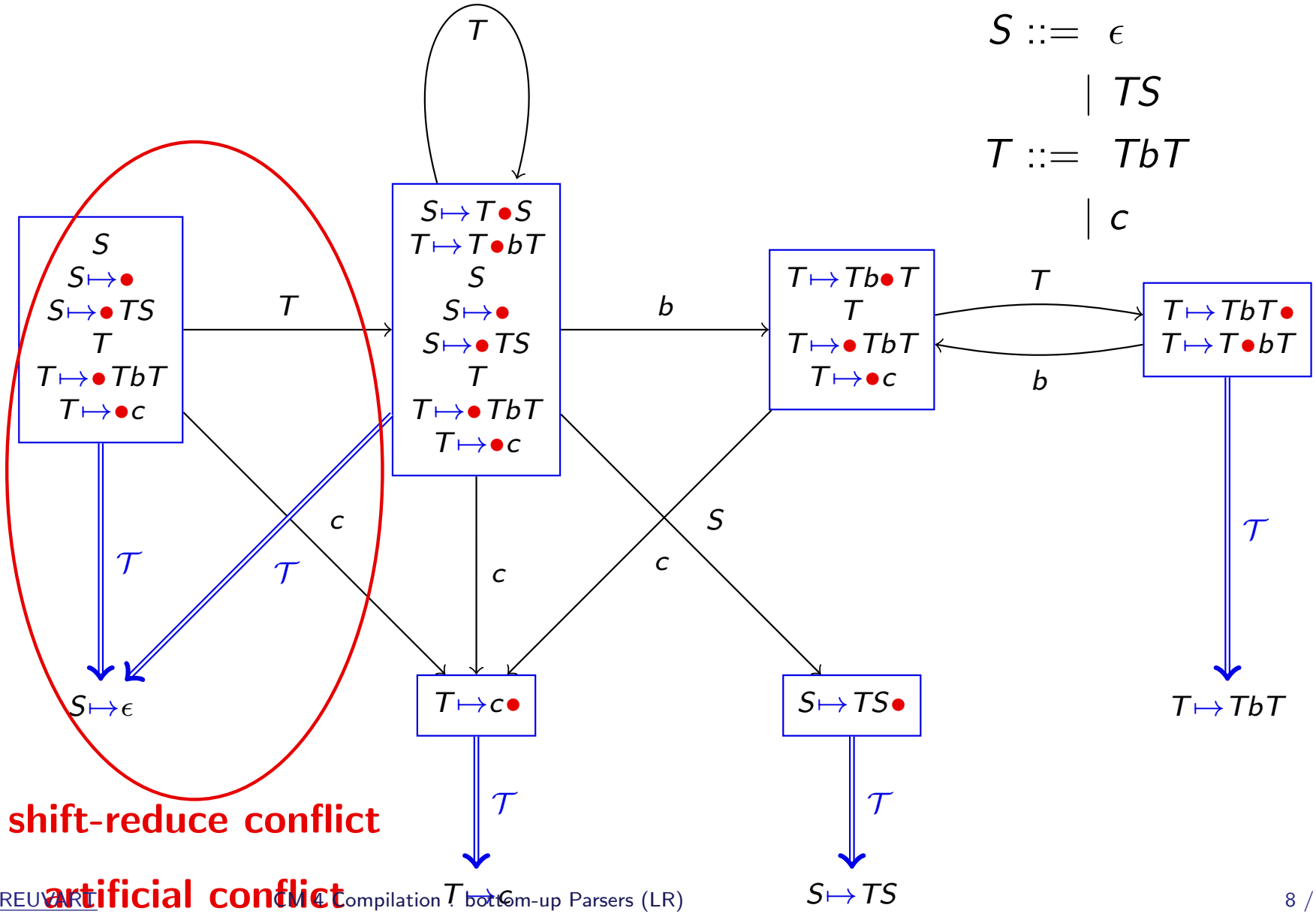
Warning, "non-ambiguous" conflicts may remain

$$S ::= \epsilon \mid TS$$
$$T ::= TbT \mid c$$



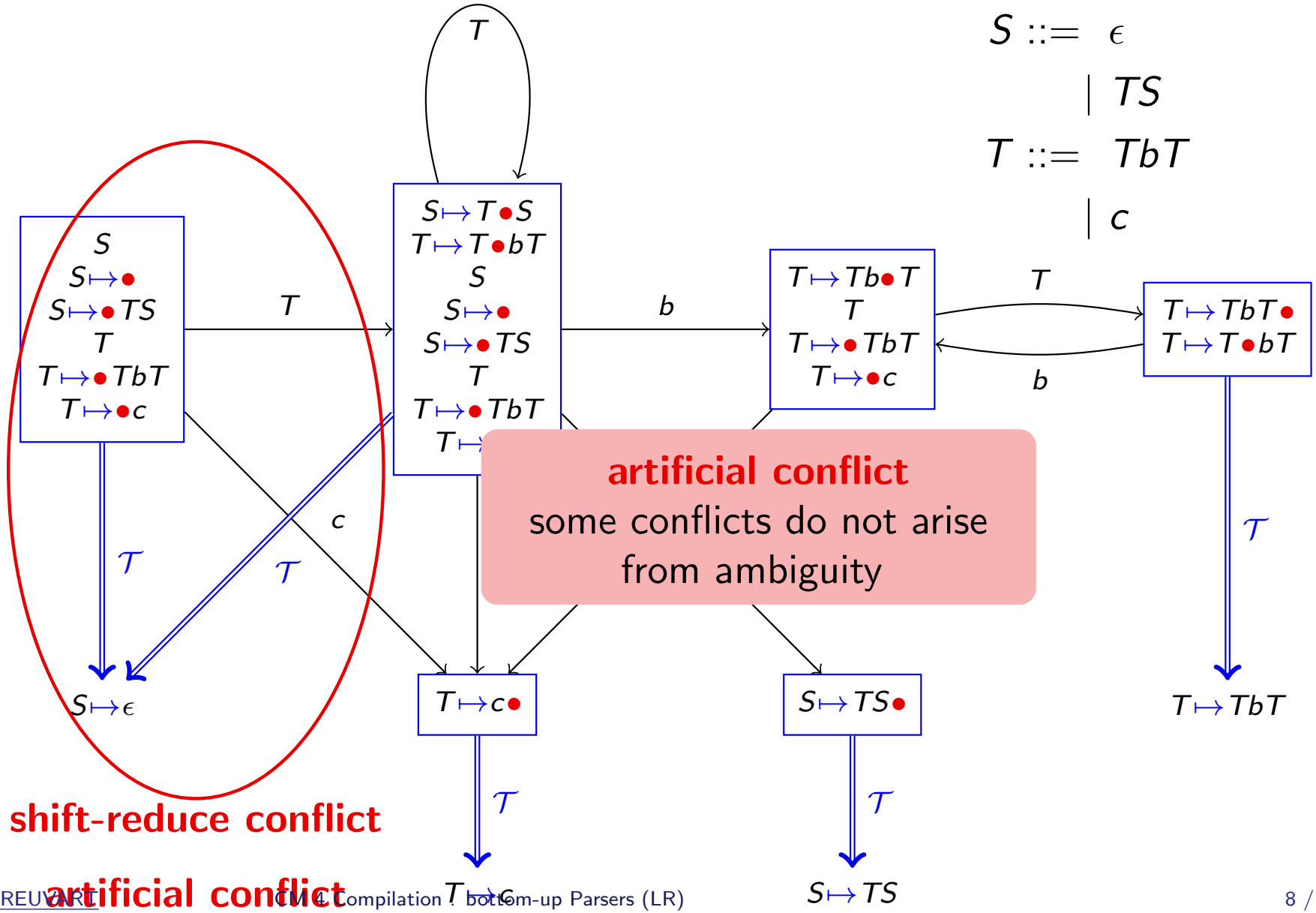
Warning, "non-ambiguous" conflicts may remain

$$S ::= \epsilon \mid TS$$
$$T ::= TbT \mid c$$



Warning, "non-ambiguous" conflicts may remain

$$S ::= \epsilon \mid TS$$
$$T ::= TbT \mid c$$



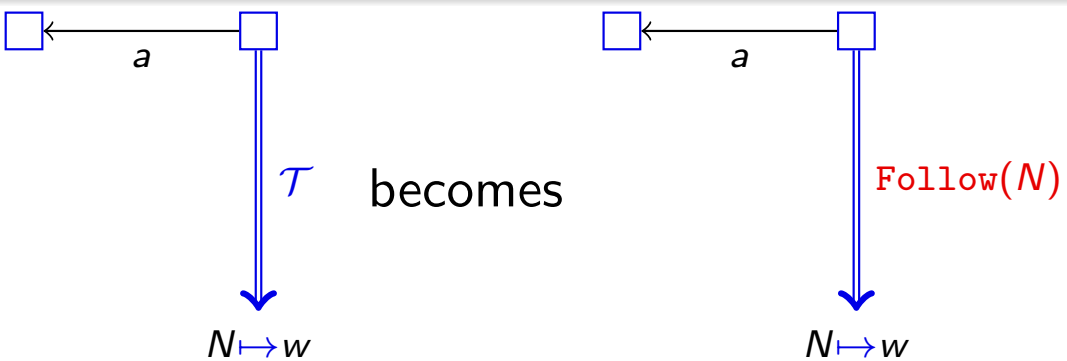
SLR Parser: narrowing conditions of actions

If a would-be NT can't be followed by peeked at terminal there is no reason to take the action

This is why we compute **"follows"** :
the set of terminals that may eventually follow a given non-terminal.

Idea : narrowing reductions paths to *Follows* only

Same construction as LR_0 except that reduction action $N \mapsto w$ are are narrowed to `Follow`(N) (before or after determinisation).



"Firsts" : terminals that may eventually start a non-terminal

Prior to "follows", let's look at easier "firsts"

Definition of $\texttt{First}_G(N) \subseteq \mathcal{T} \cup \{\varepsilon\}$

$c \in \texttt{First}_G(N) \iff$ { there exists a syntactic tree G rooted by N which leftmost terminal leaf is c.

Example with

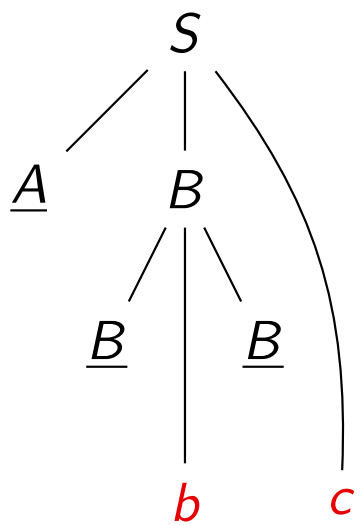
$S ::= ABc$

$A ::= \epsilon$

$\quad\quad | \; aA$

$B ::= \epsilon$

$\quad\quad | \; BbB$



Thus
$b \in \texttt{First}(S)$

Firsts without Empty Rules

Firsts: Terminals that can start a word (of NT and T) w

$$\texttt{First}(w \in (\mathcal{N} \cup \mathcal{T})^*) \subseteq \mathcal{T}$$

`First` is defined as the smallest set such that:

- if $t \in \mathcal{T}$, $\texttt{First}(t) = \{t\}$,
- for any rule $N \mapsto w$, $\texttt{First}(N) \supseteq \texttt{First}(w)$,
- $\texttt{First}(w_1 w_2) = \texttt{First}(w_1)$.

Example of *Firsts* without Empty Rules Algebraically

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$$
\begin{aligned}
&\texttt{First}(\langle exp \rangle) = \\
&\quad \texttt{First}(\langle exp \rangle \texttt{-} \langle term \rangle) \\
&\quad \cup\ \texttt{First}(\langle term \rangle) \\
&\texttt{First}(\langle term \rangle) = \\
&\quad \texttt{First}(\langle term \rangle \texttt{*} \langle fac \rangle) \\
&\quad \cup\ \texttt{First}(\langle fac \rangle) \\
&\texttt{First}(\langle fac \rangle) = \\
&\quad \texttt{First}(\texttt{(} \langle exp \rangle \texttt{)}) \\
&\quad \cup\ \texttt{First}(\texttt{-} \langle fac \rangle) \\
&\quad \cup\ \texttt{First}(\langle NB \rangle)
\end{aligned}
$$

Example of *Firsts* without Empty Rules Algebraically

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$$
\begin{aligned}
&\texttt{First}(\langle exp \rangle) = \\
&\quad \texttt{First}(\langle exp \rangle) \\
&\quad \cup\ \texttt{First}(\langle term \rangle) \\
&\texttt{First}(\langle term \rangle) = \\
&\quad \texttt{First}(\langle term \rangle) \\
&\quad \cup\ \texttt{First}(\langle fac \rangle) \\
&\texttt{First}(\langle fac \rangle) = \\
&\quad \texttt{First}(\texttt{(}) \\
&\quad \cup\ \texttt{First}(\texttt{-}) \\
&\quad \cup\ \texttt{First}(\langle NB \rangle)
\end{aligned}
$$

Example of *Firsts* without Empty Rules Algebraically

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$$
\begin{aligned}
&\texttt{First}(\langle exp \rangle) = \\
&\quad \texttt{First}(\langle exp \rangle) \\
&\quad \cup\ \texttt{First}(\langle term \rangle) \\
&\texttt{First}(\langle term \rangle) = \\
&\quad \texttt{First}(\langle term \rangle) \\
&\quad \cup\ \texttt{First}(\langle fac \rangle) \\
&\texttt{First}(\langle fac \rangle) = \\
&\quad \texttt{First}(\texttt{(}) \\
&\quad \cup\ \texttt{First}(\texttt{-}) \\
&\quad \cup\ \texttt{First}(\langle NB \rangle)
\end{aligned}
$$

Example of *Firsts* without Empty Rules Algebraically

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$$
\begin{aligned}
&\texttt{First}(\langle exp \rangle) = \\
&\quad \texttt{First}(\langle exp \rangle) \\
&\quad \cup\ \texttt{First}(\langle term \rangle)
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{First}(\langle term \rangle) = \\
&\quad \texttt{First}(\langle term \rangle) \\
&\quad \cup\ \texttt{First}(\langle fac \rangle)
\end{aligned}
$$

$$\texttt{First}(\langle fac \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

Example of *Firsts* without Empty Rules Algebraically

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$$
\begin{aligned}
&\texttt{First}(\langle exp \rangle) = \\
&\quad \texttt{First}(\langle exp \rangle) \\
&\quad \cup\ \texttt{First}(\langle term \rangle)
\end{aligned}
$$

$$\texttt{First}(\langle term \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

$$\texttt{First}(\langle fac \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

Example of *Firsts* without Empty Rules Algebraically

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$$\texttt{First}(\langle exp \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

$$\texttt{First}(\langle term \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

$$\texttt{First}(\langle fac \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

Example of *Firsts* without Empty Rules By Fixed Point

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$\texttt{First}(\langle exp \rangle) \supseteq \{\}$

$\texttt{First}(\langle term \rangle) \supseteq \{\}$

$\texttt{First}(\langle fac \rangle) \supseteq \{\}$

Example of *Firsts* without Empty Rules By Fixed Point

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$\texttt{First}(\langle exp \rangle) \supseteq \{\}$

$\texttt{First}(\langle term \rangle) \supseteq \{\}$

$\texttt{First}(\langle fac \rangle) \supseteq \{\texttt{(}\}$

Example of *Firsts* without Empty Rules By Fixed Point

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$\texttt{First}(\langle exp \rangle) \supseteq \{\}$

$\texttt{First}(\langle term \rangle) \supseteq \{\}$

$\texttt{First}(\langle fac \rangle) \supseteq \{\texttt{(}, \texttt{-}\}$

Example of *Firsts* without Empty Rules By Fixed Point

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$\texttt{First}(\langle exp \rangle) \supseteq \{\}$

$\texttt{First}(\langle term \rangle) \supseteq \{\}$

$\texttt{First}(\langle fac \rangle) \supseteq \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$

Example of *Firsts* without Empty Rules By Fixed Point

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$\texttt{First}(\langle exp \rangle) \supseteq \{\}$

$\texttt{First}(\langle term \rangle) \supseteq \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$

$\texttt{First}(\langle fac \rangle) \supseteq \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$

Example of *Firsts* without Empty Rules By Fixed Point

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$\texttt{First}(\langle exp \rangle) \supseteq \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$

$\texttt{First}(\langle term \rangle) \supseteq \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$

$\texttt{First}(\langle fac \rangle) \supseteq \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$

Example of *Firsts* without Empty Rules By Fixed Point

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Firsts

$$\texttt{First}(\langle exp \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

$$\texttt{First}(\langle term \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

$$\texttt{First}(\langle fac \rangle) = \{\texttt{(}, \texttt{-}, \langle NB \rangle\}$$

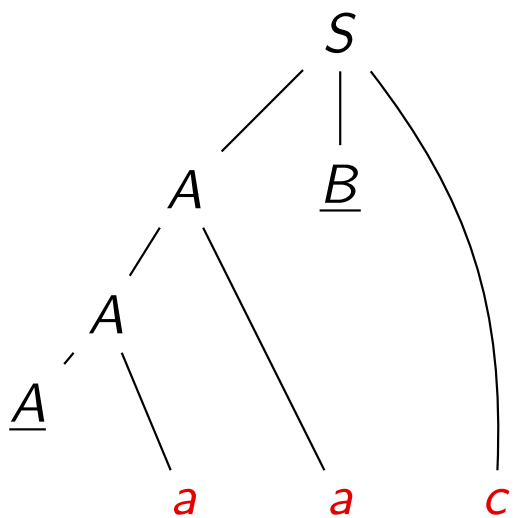
"Follows" : terminals that may eventually follow a non-terminal.

Definitions of $\texttt{Follow}_G(N) \subseteq \sqcup \cup \{\$\}$

$c \in \texttt{Follow}_G(N) \iff$ { There exists a ST G with an internal node N which leftmost terminal leas RIGHT OF N is c.

Example ins

$$S ::= ABc$$
$$A ::= \epsilon \mid aA$$
$$B ::= \epsilon \mid BbB$$



Thus
$a \in \texttt{Follow}(A)$
$c \in \texttt{Follow}(A)$

Follows

Follows : the terminals that can follow a non-terminal N

The symbol \$ is used to denote the fact that N can be at the end of a file

$$\texttt{Follow}(N \in \mathcal{N}) \subseteq \mathcal{T} \cup \{\$\}$$

`Follow` is defined by the smallest set such that :

- if S is the main NT,
$$\texttt{Follow}(S) \ni \$ \ ,$$
- for each rule $N' \mapsto w_1 N w_2$,
$$\texttt{Follow}(N) \supseteq \texttt{First}(w_2) \setminus \{\epsilon\} \cup \texttt{Follow}(N')$$
- for each rule $N' \mapsto w_1 N$,
$$\texttt{Follow}(N) \supseteq \texttt{Follow}(N')$$

Example of *Follow* (without ϵ rule)

Grammar

$$\langle exp \rangle ::= \langle exp \rangle \texttt{-} \langle term \rangle \mid \langle term \rangle$$

$$\langle term \rangle ::= \langle term \rangle \texttt{*} \langle fac \rangle \mid \langle fac \rangle$$

$$\langle fac \rangle ::= \texttt{(} \langle exp \rangle \texttt{)} \mid \texttt{-} \langle fac \rangle \mid \langle NB \rangle$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\}$

Example of *Follow* (without ϵ rule)

Grammar

$$\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle - \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle * \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= (\langle exp \rangle) \\
&\mid -\langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\}$

Example of *Follow* (without ϵ rule)

Grammar

$$\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$, \texttt{-}\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\}$

Example of *Follow* (without ϵ rule)

Grammar

$$\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$, \texttt{-}\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\$, \texttt{-}\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\}$

Example of *Follow* (without ϵ rule)

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$, \texttt{-}\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\}$

Example of *Follow* (without ϵ rule)

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= (\langle exp \rangle) \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$, \texttt{-}\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}\}$

Example of *Follow* (without ϵ rule)

Grammar

$$\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$, \texttt{-}, \texttt{)}\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}\}$

Example of *Follow* (without ϵ rule)

Grammar

$$\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$, \texttt{-}, \texttt{)}\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}, \texttt{)}\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}\}$

Example of *Follow* (without ϵ rule)

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Follows

$\texttt{Follow}(\langle exp \rangle) \supseteq \{\$, \texttt{-}, \texttt{)}\}$

$\texttt{Follow}(\langle term \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}, \texttt{)}\}$

$\texttt{Follow}(\langle fac \rangle) \supseteq \{\$, \texttt{-}, \texttt{*}, \texttt{)}\}$

Example of *Follow* (without ϵ rule)

Grammar

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle \\
\langle term \rangle &::= \langle term \rangle \texttt{*} \langle fac \rangle \\
&\mid \langle fac \rangle \\
\langle fac \rangle &::= \texttt{(} \langle exp \rangle \texttt{)} \\
&\mid \texttt{-} \langle fac \rangle \\
&\mid \langle NB \rangle
\end{aligned}
$$

Follows

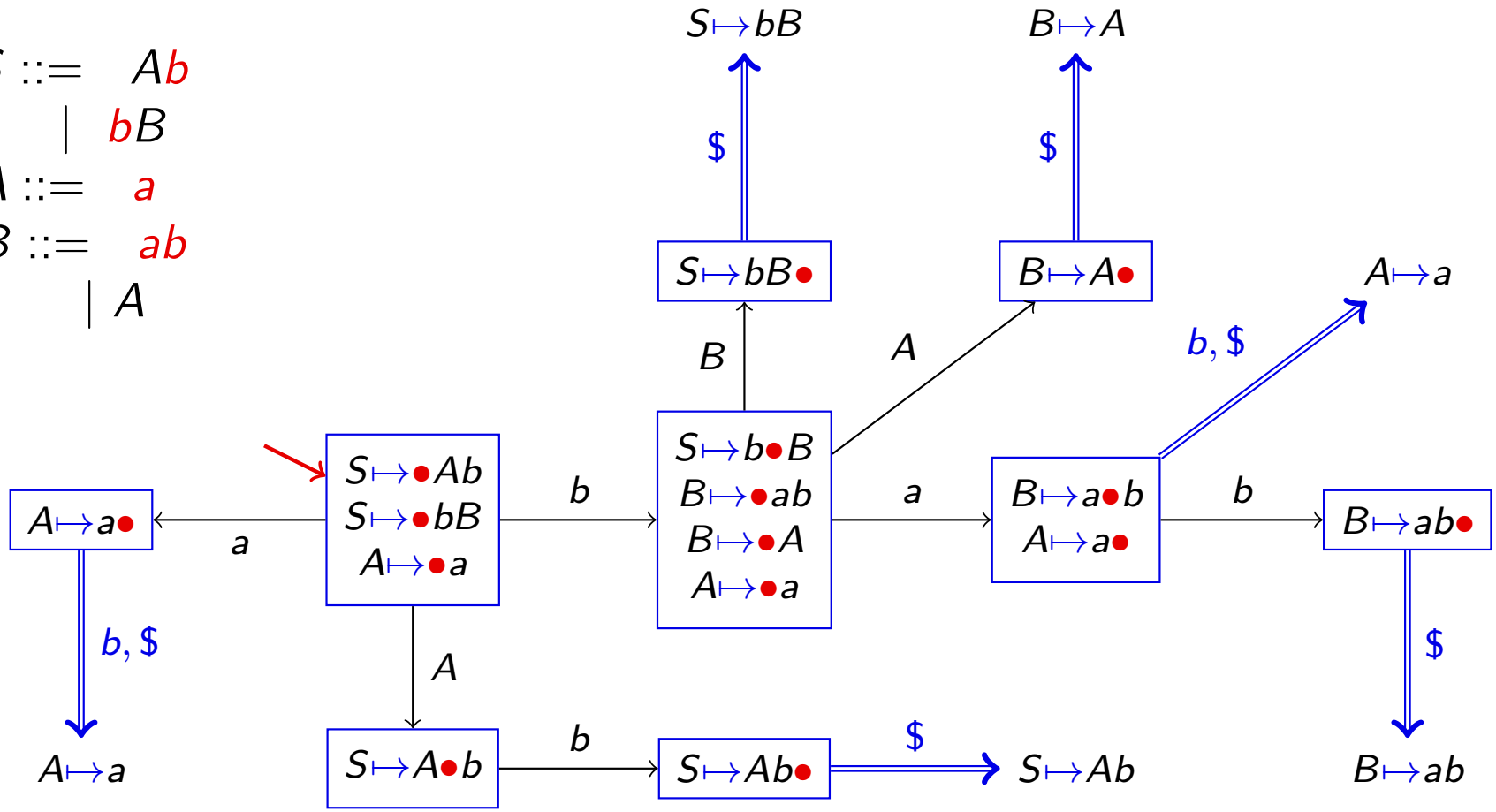
$\texttt{Follow}(\langle exp \rangle) = \{\$, \texttt{-}, \texttt{)}\}$

$\texttt{Follow}(\langle term \rangle) = \{\$, \texttt{-}, \texttt{*}, \texttt{)}\}$

$\texttt{Follow}(\langle fac \rangle) = \{\$, \texttt{-}, \texttt{*}, \texttt{)}\}$

Resolves many conflicts but not all...

$S ::= Ab \mid bB$
$A ::= a$
$B ::= ab \mid A$



Infinite headlong rush (LALR,LR1,LR2...)

Capturing all non-ambiguous grammar without creating conflict is impossible

The question of the ambiguously of a grammar is not decidable.

In practice: LALR and LR1 (for prog. languages)

LALR and LR1 parsers are more powerful and with reasonable overcost.

How to see your parser automaton in practice ?

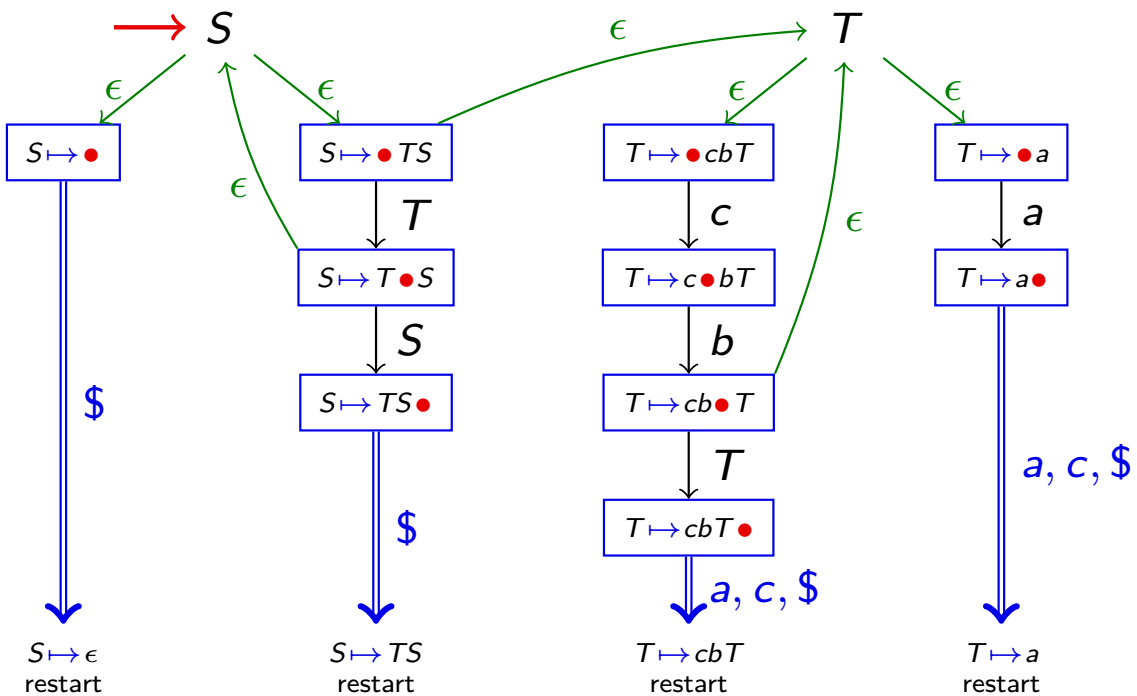
Using Bison, Yacc or CamlYacc

- Compile with `-v` option,
- the result is in `y.output`,
- (using `-gv` option you can have a .dot graph)

What about Java ???

Idea : Execute the ND automaton using a deterministic oracle using firsts and follow

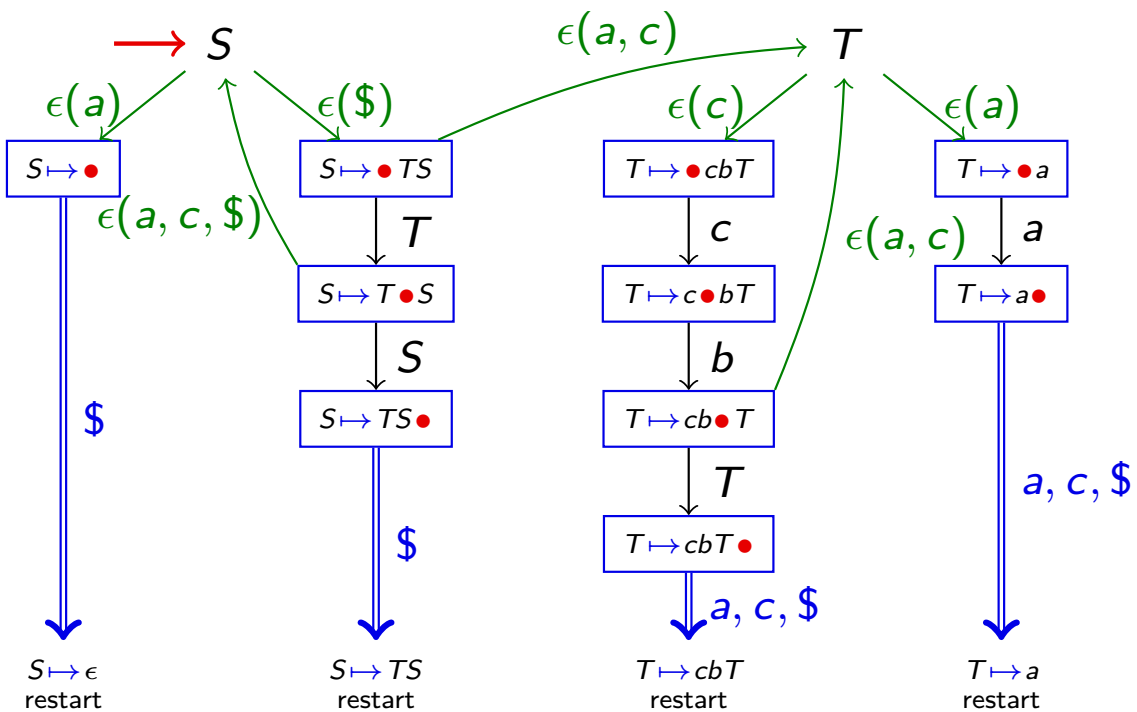
- replace ϵ-transitions by "peeks" (or "look-aheads" – the next letter is red but not consumed),
- utilize firsts and follows to determine these look-aheads.



What about Java ???

Idea : Execute the ND automaton using a deterministic oracle using firsts and follow

- replace ϵ-transitions by "peeks" (or "look-aheads" – the next letter is red but not consumed),
- utilize firsts and follows to determine these look-aheads.



More precisely

Change of paradigm

This is not seen as an automaton anymore, but as mutually recursive functions (one for each non terminal).

Principle of LL_1

The parsing function of N will

- look at the first terminal t,
- "guess" the rule $N \mapsto w$ as "the one" such that $t \in \texttt{First}(w) \textcircled{\in} \texttt{Follow}(N)$,
- successively call parsers for each element of the pattern.

Principle of LL_*

Idem but can extend the 'lookahead" to be able to

Pseudo-code ≃ generated code of TP1.ex1

```
public void expression() { terme();
   while (true) { switch (word.peak()) {
    case '+' : word.pop();term();
    case '-' : word.pop();term();
    case '\n',')' : return;
    default : raise ParseError;
} } }
public void term() { factor();
  while (true) { switch (word.peak()) {
    case '*' : word.pop();factor();
    case '\n','+','-',')' : return;
    default : raise ParseError;
} } }
public void factor() {
  switch (word.peak()) {
    case '(' : word.pop();expression();assert(word.pop()==')');
    case '-' : word.pop();facte r();
    case <NOMBRE> : word.pop();
    default : raise ParseError;
} }
```

Guessing using Firsts and Follows

Reminder : in Java, a non-terminal is mapped to a regexp of terminals+NT

On cases disjunction

These cases correspond to Firsts of each pattern
ex : $\texttt{First}(+\langle\texttt{term}\rangle) = \{+\}$

On Kleene star (or recursive non-terminal)

Idem plus an additional case to exit the loop using a Follow.
ex : $\texttt{Follow}(\langle\texttt{term}\rangle) = \{\backslash n, +, -,)\}$

Conflict

If these are two conflicting "cases" in a switch then the grammar is not considered LL_1

Limiters of LL grammars

The rule used for a node is guessed using the first token only

As opposite to SLR that wait for the last token.
These forces the grammar to be very procedural.

LL_*

The programmer can locally augment the "lookahead" to look for more tokens, but it require him acting, and it do not solve anything if you need the last one...

Left recursion (thus left associativity) is impossible

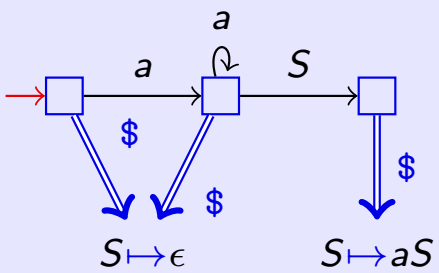
$S := Sa|b$ needs to look for a non-bounded number of elements ahead.

(Solved by using a right associativity and correct it in the actions)

Optimization : linear time

The constructed SLR automaton can be of quadratic complexity

The grammar $S := aS|\epsilon$ produce the automaton :



Links to literature

Stack automata

The state idea are pushed into a stack
$\rightsquigarrow$ Stack automata are required to recognize a grammar without backtracking

LR "GoTo"s = non-terminal shifts

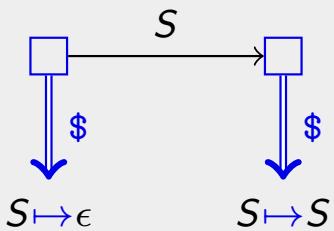
The shift/goto table is the automaton table.

A "shift" labeled by a non-terminals is called "goto".

A small bug remains : risk of looping

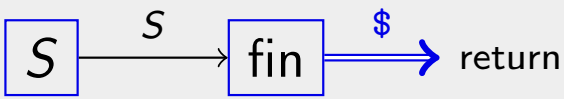
The execution may loop without conflict

The grammar $S := \epsilon | S$ produce the automaton:



Solution

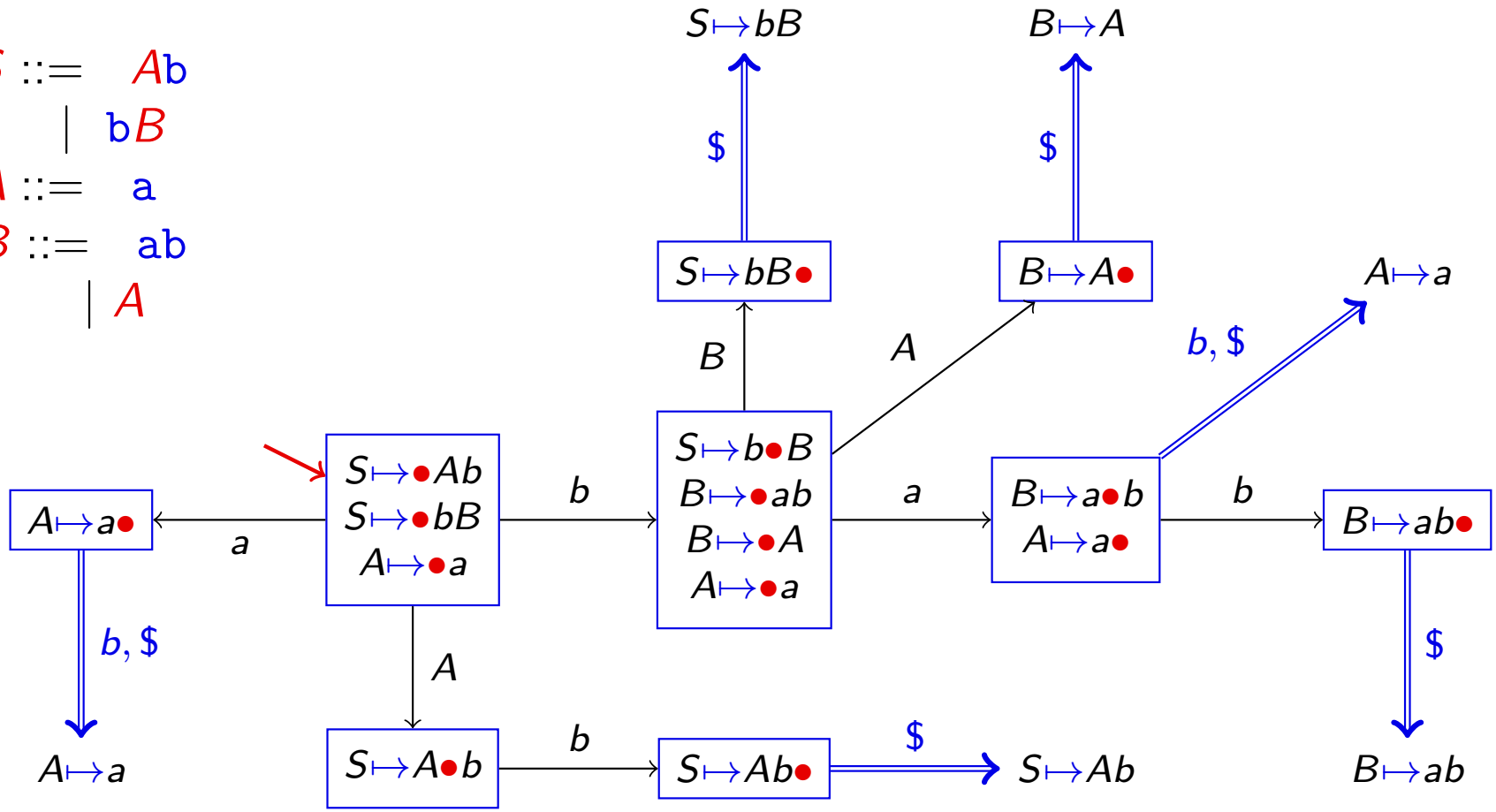
Adding, in the deterministic automaton, a virtual non terminal :



This will introduce an action-action conflict in place of the loop (resolvable by adding a priority).

Some artificial conflicts remaining

$S ::= Ab \mid bB$
$A ::= a$
$B ::= ab \mid A$



LR_1 Automaton: a distinct non-terminal for each follow

The grammar is refined by duplicating non-terminals

- For each $N \in \mathcal{T}$ and each $t \in \texttt{Follow}(N)$ we create N_t,
- Rules of N_t are those of N, but split for each encountered non-terminal
- Compute the SLR automaton on resulting grammar,
- Remove annotations.

$$
\begin{aligned}
S &::= A\texttt{b} \\
&\mid \texttt{b}B \\
A &::= \texttt{a} \\
B &::= \texttt{ab} \\
&\mid A
\end{aligned}
$$

becomes

$$
\begin{aligned}
S_{\$} &::= A_b\texttt{b} \\
&\mid \texttt{b}B_{\$} \\
A_{\$} &::= \texttt{a} \\
A_b &::= \texttt{a} \\
B_{\$} &::= \texttt{ab} \\
&\mid A_{\$}
\end{aligned}
$$

Computing intermediate grammar for LR_1 algorithm

Reminder:$\mathcal{G} = (\mathcal{T}, \mathcal{N}, \mathcal{R})$

$\mathcal{T}$: terminlks $\mathcal{N}$: non terminals
$\mathcal{R} \subseteq \mathcal{N} \times (\mathcal{N}+\mathcal{T})^*$ rules of the form $N \mapsto w$.

$\texttt{LR1}(\mathcal{G}) = (\mathcal{T}, \mathcal{N}', \mathcal{R}'$

$$\mathcal{N}' = \{N_t \mid N \in \mathcal{N}, t \in \texttt{Follow}(N)\}$$

$$\mathcal{R}' = \{N_t \mapsto w' \mid (N \mapsto w) \in \mathcal{R},\ w' \in \langle w \mid t \rangle\}$$

$$\langle wt' \mid t \rangle = \langle w \mid t' \rangle t'$$

$$\langle wN \mid t \rangle = \bigcup_{t' \in \texttt{First}(Nt)} \langle w \mid t' \rangle N_t$$

LR_1 parser are huge

$$
\begin{array}{l}
S ::= AbB \\
\quad\quad | \; bAB \\
A ::= Ba \\
B ::= aS \\
\quad\quad | \; bA
\end{array}
\quad \text{devient} \quad
\begin{array}{l}
S_\$ ::= A_b b \\
\quad\quad | \; bA_a B_\$ \\
\quad\quad | \; bA_b B_\$ \\
A_\$ ::= B_a a \\
A_a ::= B_a a \\
B_\$ ::= ab \\
\quad\quad | \; bA_\$
\end{array}
\quad
\begin{array}{l}
S_a ::= A_b b \\
\quad\quad | \; bA_a B_a \\
\quad\quad | \; bA_b B_a \\
A_b ::= B_a a \\
\\
B_a ::= ab \\
\quad\quad | \; bA_a
\end{array}
$$

The resulting grammar has its size multiplied by T^p

where T it the number of terminals and p the maximum number of terminals appearing in a single rule.
... and this is before determinisation which is potentially exponential.

Used in modern generators

Minimized version

Minimisation can be peformed on the fly, resulting in re more reasonably sized parser.

Modern LR parser generators use LR_1.

They are not seen in TP because they are more dificult to install.

LR_k

It is possible to perform a LR algorithms splitting the grammar using more "look-ahead" information (similar to a follow but looking for sub-words of size k rather than letters), the resulting algorithm is called LR_k.

The resulting grammar becomes of size

$$T^{p*k}$$

which soon becomes unrealistic, for little gain.

Universality of LR_1

As we have seen, any grammair LR_k can be modified into an "equivalent" *SLR* grammar.
In fact any non-ambiguous grammar can be modified into a *SLR* grammar, but there is no generic algorithm and there can't be any (the proof is fundamentally non-constructive).

LALR : A SLR-sized parser nearly as expressive as LR_1

Principle

- Create a psedo-deterministic LR_1 parser
- errase annotation,
- fuse states with the same names

The fusion will not create new shift-action conflict

Because shifts are forced by names.

Howeover, there could be new action-action conflicts (corresponding to the creation of the same non-terminals from diferent follows).

Size of SLR

Since the names are set of the the non-deterministic LR_0

Used by "old" parser generators such as Bison or Yacc.

Why not using a unic algorithm

Ambiguity is not decidable

It is only recursivelly enumerable, which mean that it is alway possible to have a proof of ambiguity, but not necessarlily the other way around.

Prouvable via a reduction to "Post correspondance" problem.

Consequence: to accept any non-ambiguous grammar, we have to accept any (context-free) grammar.

Not interesting for programming language: we need to be sure that there is no ambuguity in order not to confuse the programmer.
In addition those are longuer to parse.

Chart parsers : GLR, Earley, CYK, Packrat...

Those are "universal" parser generators that accept any (context-free) grammar, but which parsers can produce several trees from the same input.

Generated parsers work in time $o(n^{2+\epsilon})$

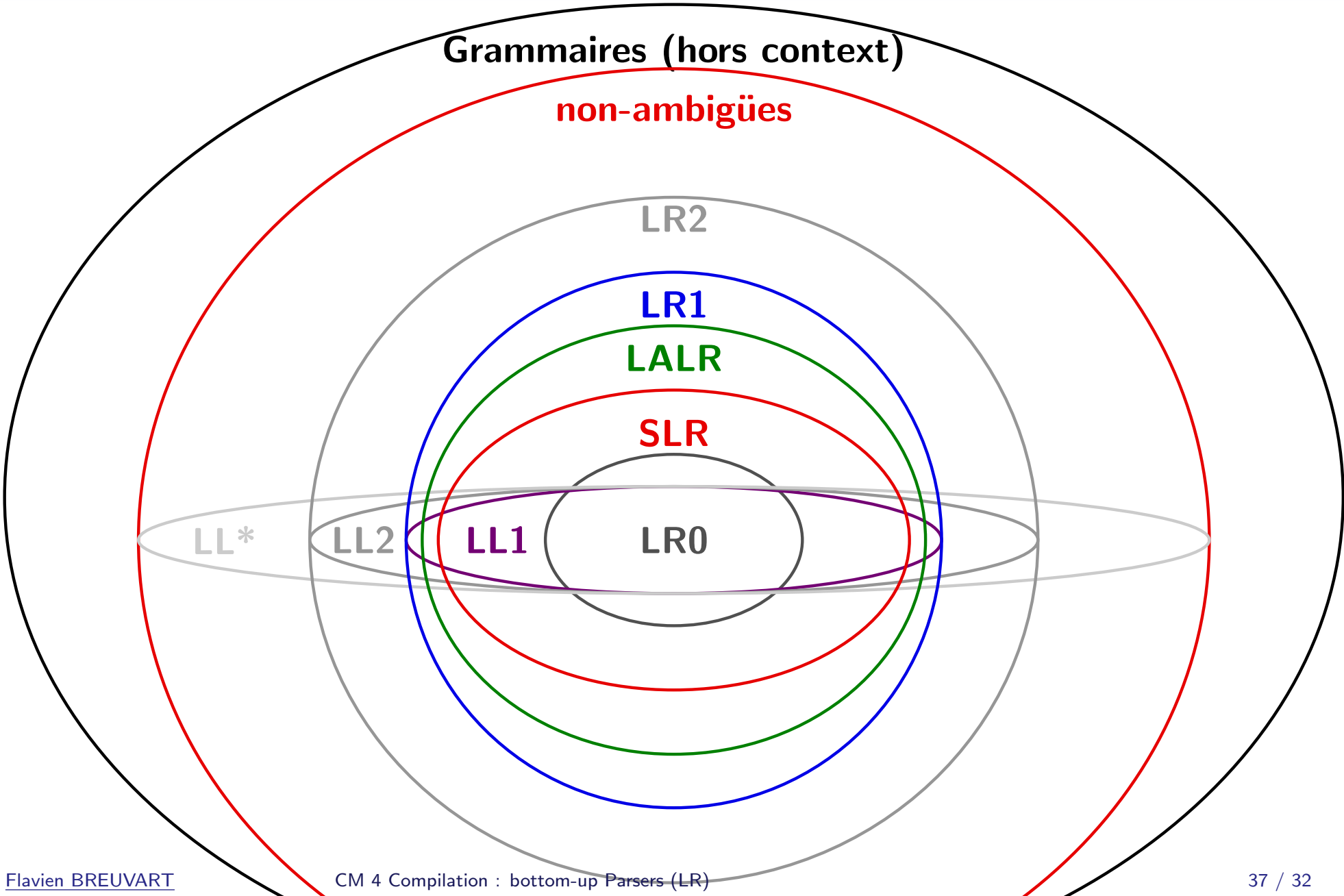
When the grammar in non-ambiguous, they are linear, but with arbitrarily large constant.

Same coplexity as the matrix multiplication

Used for natural language processing

Human language are all ambiguous.

Potatoes view of parser generators



Exercise

$$
\begin{aligned}
\langle expr \rangle ::= & \langle INT \rangle \\
& \mid \langle IValue \rangle \\
& \mid \langle IValue \rangle ++ \\
& \mid \langle expr \rangle \&\& \langle expr \rangle \\
& \mid \langle expr \rangle + \langle expr \rangle \\
& \mid \langle expr \rangle \langle expr \rangle \\
\langle IValue \rangle ::= & \langle IDEN \rangle \\
& \mid \langle expr \rangle [\langle expr \rangle] \\
& \mid \langle expr \rangle *
\end{aligned}
$$