CM 4 Compilation : bottom-up Parsers (LR)

Flavien BREUVART

February 12, 2026

Parseur LR in motion

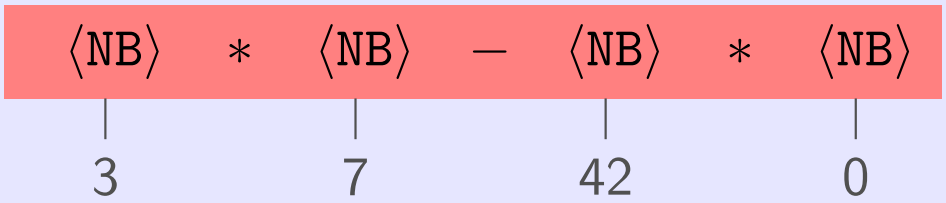
The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

Grammar :

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

Reconstructing the tree :



Parseur LR in motion

The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

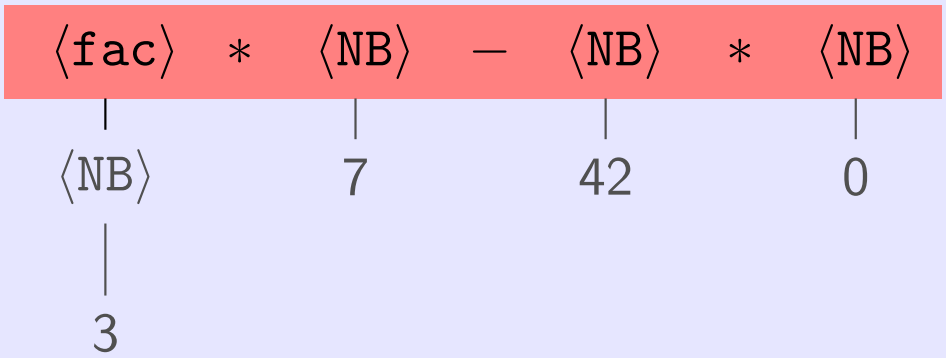
Grammar :

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

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

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

Reconstructing the tree :



Parseur LR in motion

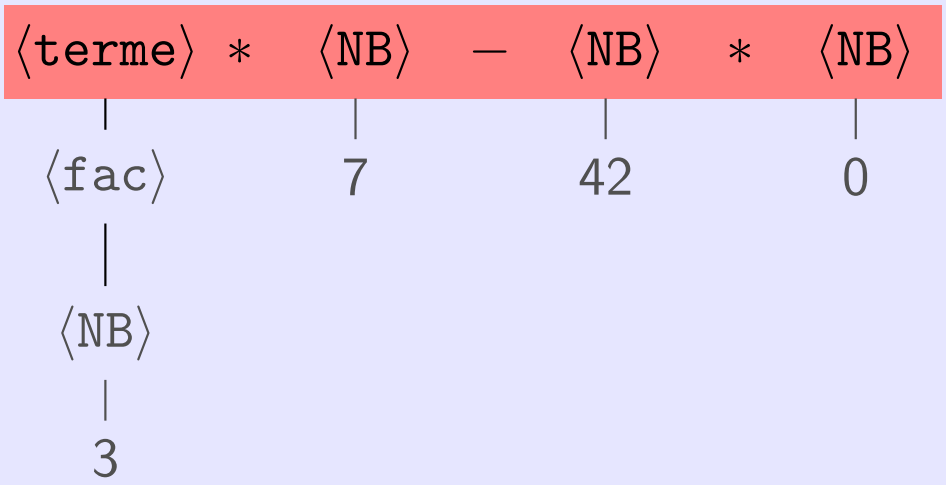
The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

Grammar :

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

Reconstructing the tree :



Parseur LR in motion

The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

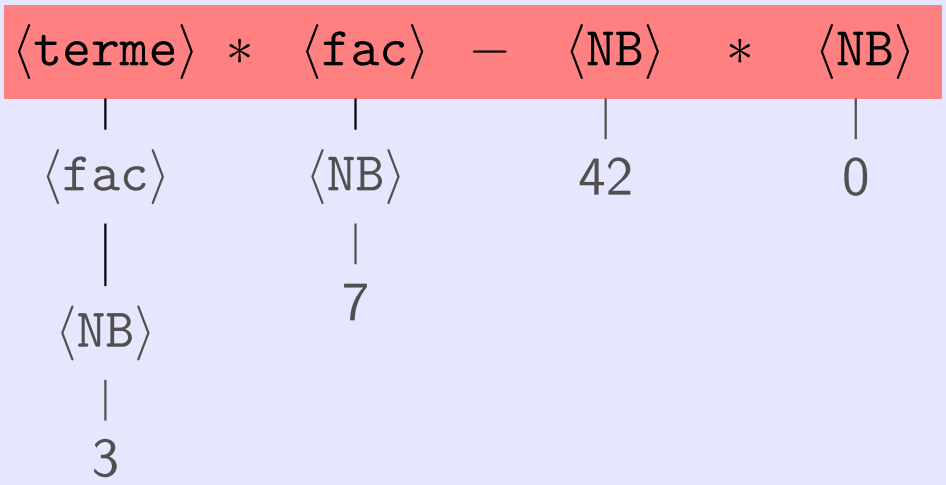
Grammar :

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

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

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

Reconstructing the tree :



Parseur LR in motion

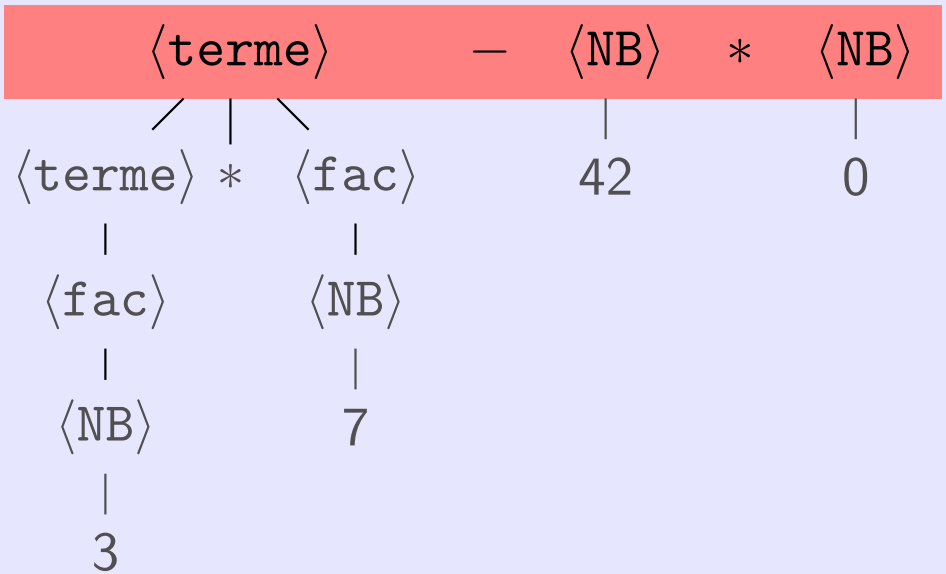
The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

Grammar :

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

Reconstructing the tree :



Parseur LR in motion

The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

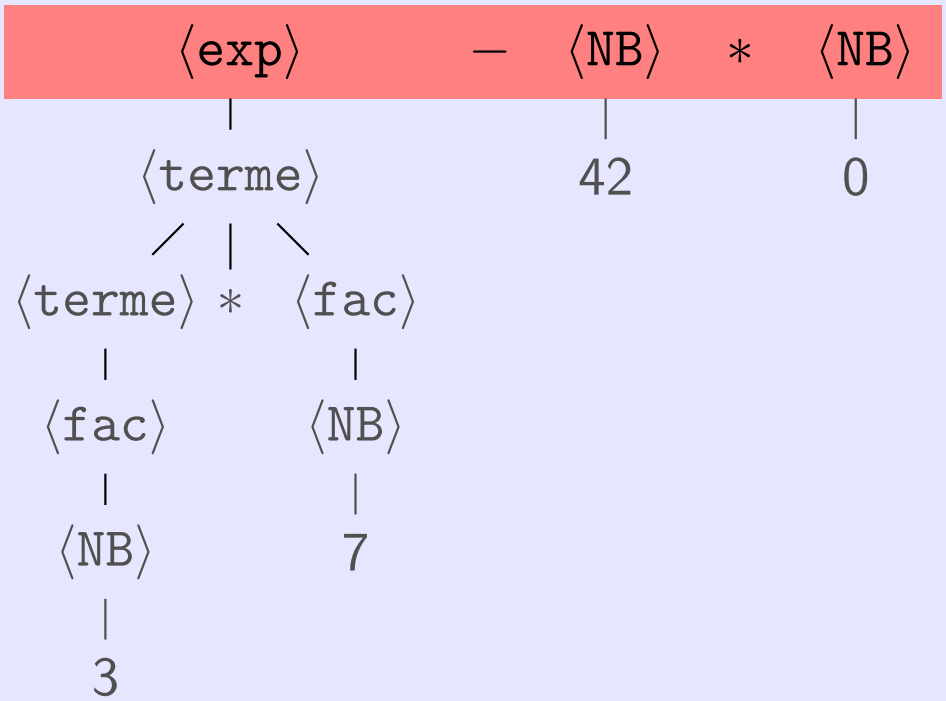
Grammar :

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

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

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

Reconstructing the tree :



Parseur LR in motion

The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

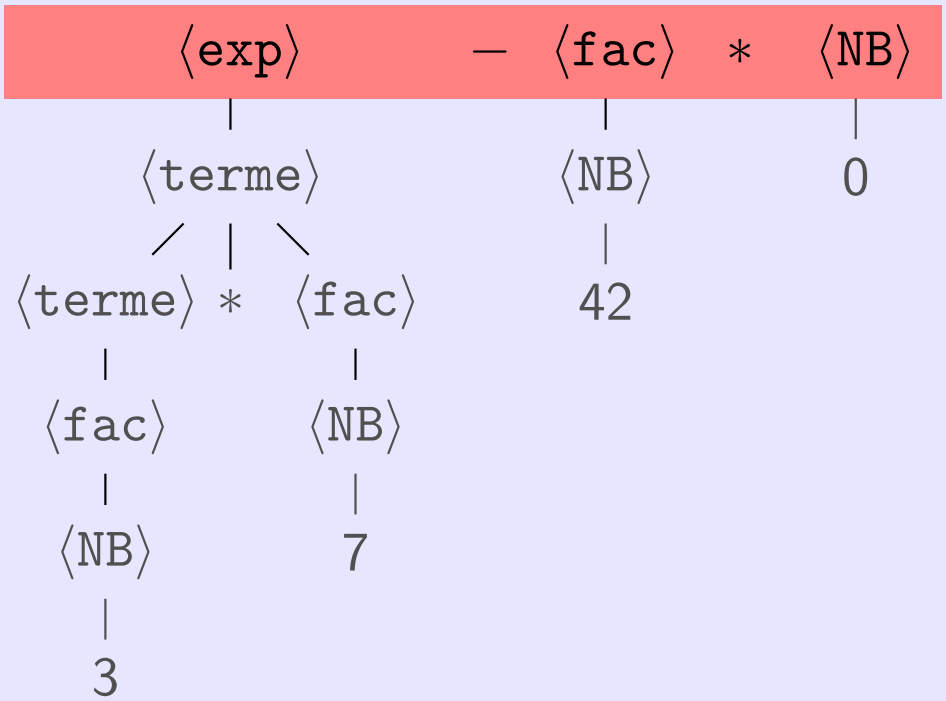
Grammar :

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

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

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

Reconstructing the tree :



Parseur LR in motion

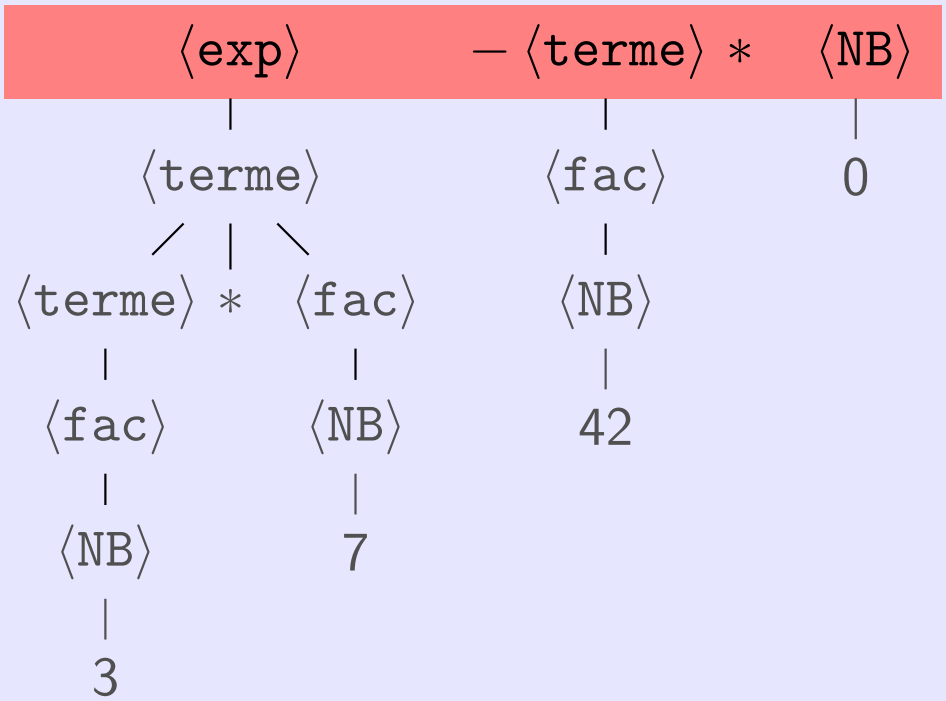
The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

Grammar :

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

Reconstructing the tree :



Parseur LR in motion

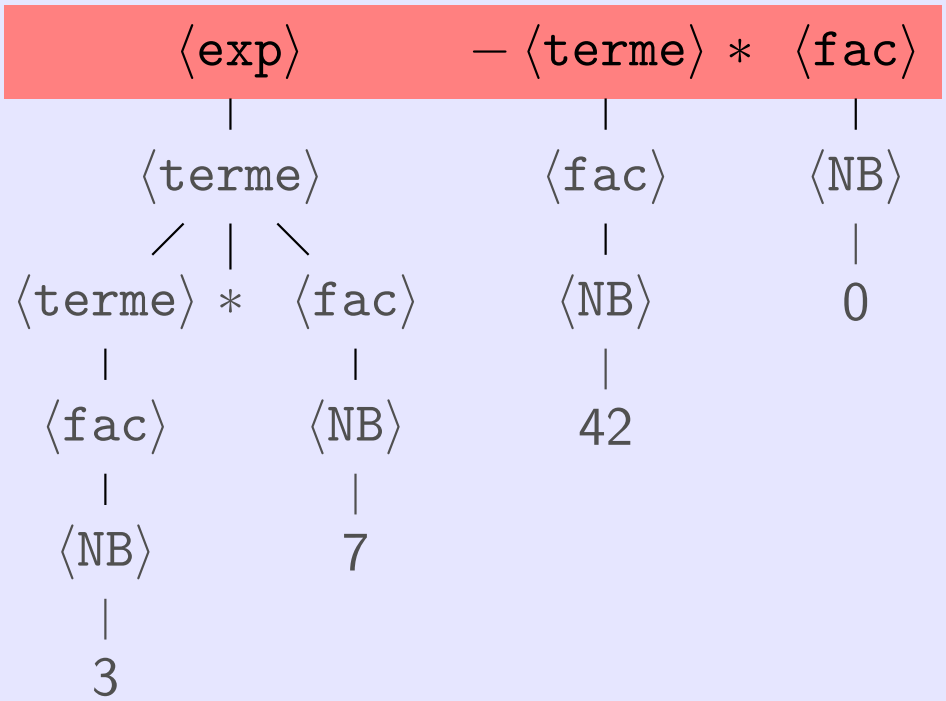
The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

Grammar :

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

Reconstructing the tree :



Parseur LR in motion

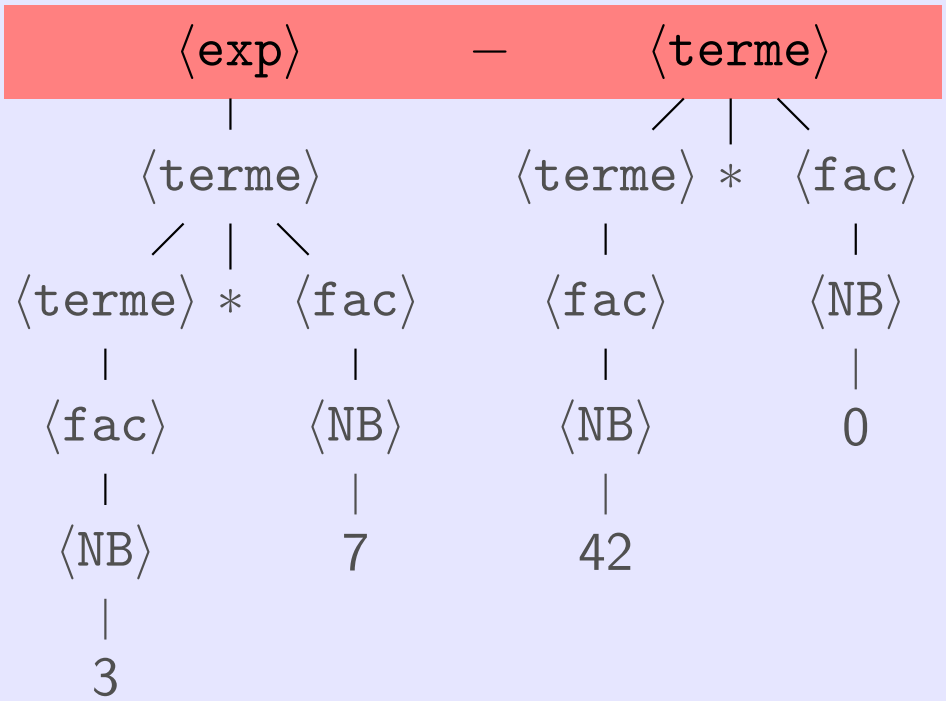
The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

Grammar :

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

Reconstructing the tree :



Parseur LR in motion

The parser only read the names of constructed tokens

On a ST, it only looks at the root label.

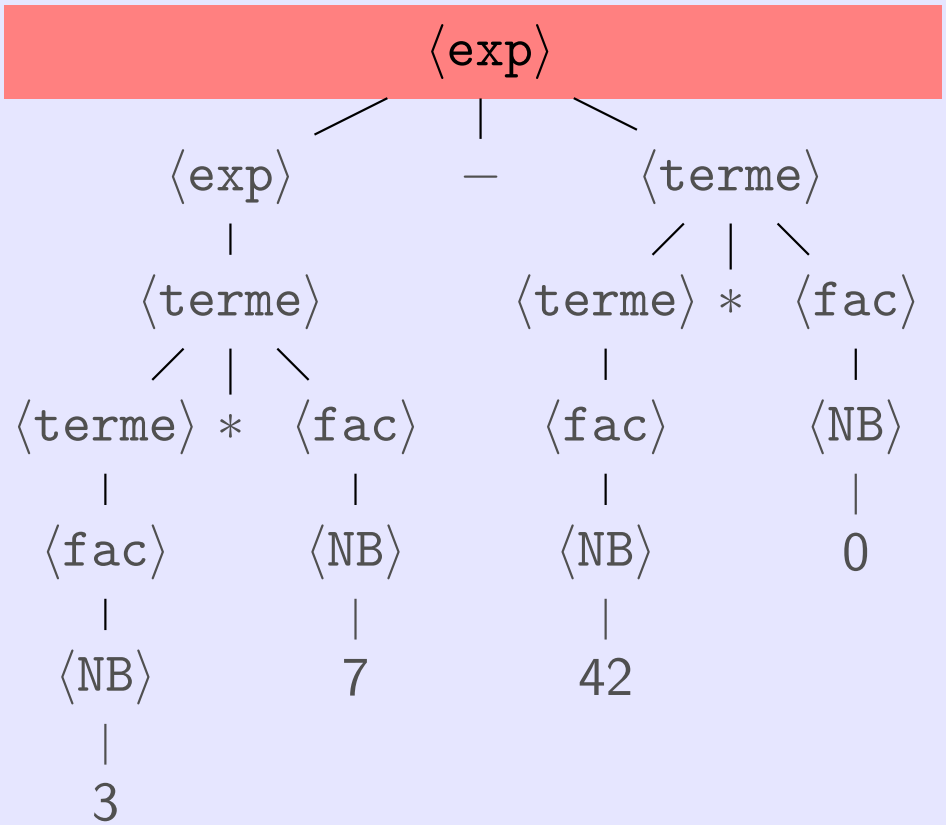
Grammar :

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

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

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

Reconstructing the tree :



Concept of the non-deterministic parser

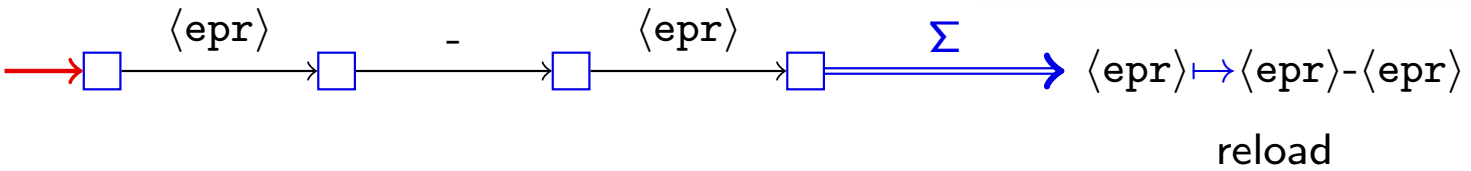
Idea : write a non-deterministic shift-action automaton

Shifts :

- read a word of terminal and non terminals
- look for the pattern of a rule.

Action :

- apply the uncovered rule,
- create a node,
- reload the automaton (begining).



Remark : Σ remplace `Char`

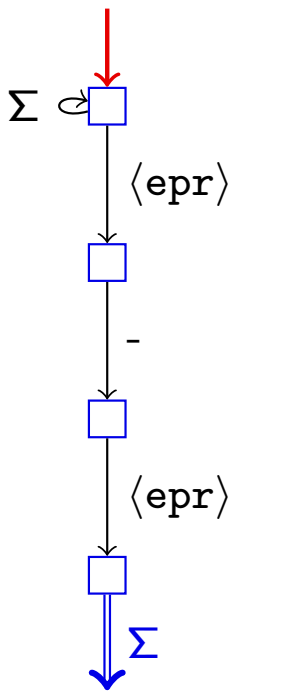
$\Sigma = \mathcal{T} \cup \mathcal{N}$ is the set of terminals + non-terminals

First attempt

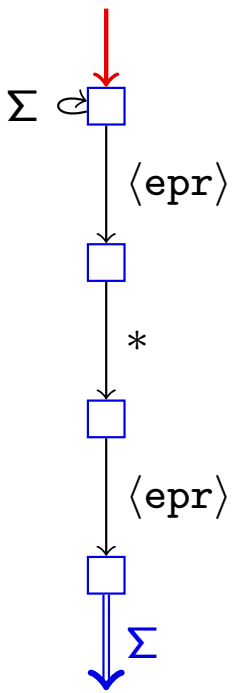
Idea : Same as for the lexer

- a small automaton for each rule that "test" the patern,
- preceded by a delaying loop,
- adds actions and takes the sum.

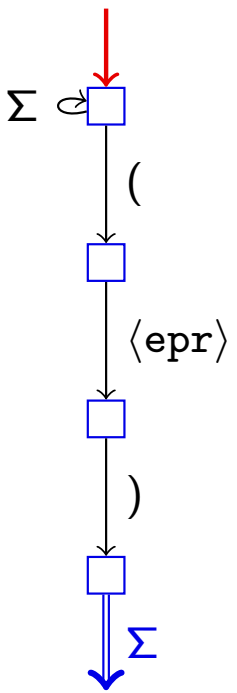
Remark : such an action is called reduct



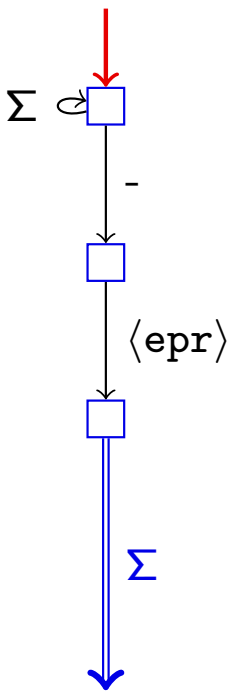
⟨epr⟩↦⟨epr⟩-⟨epr⟩
relancer



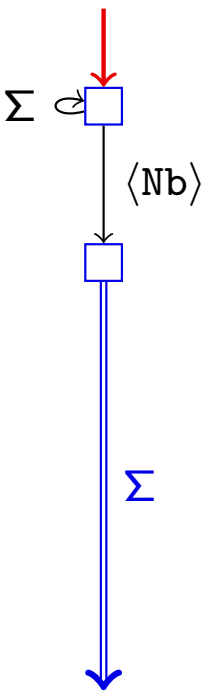
⟨epr⟩↦⟨epr⟩*⟨epr⟩
relancer



⟨epr⟩↦(⟨epr⟩)
relancer



⟨epr⟩↦-⟨epr
relancer



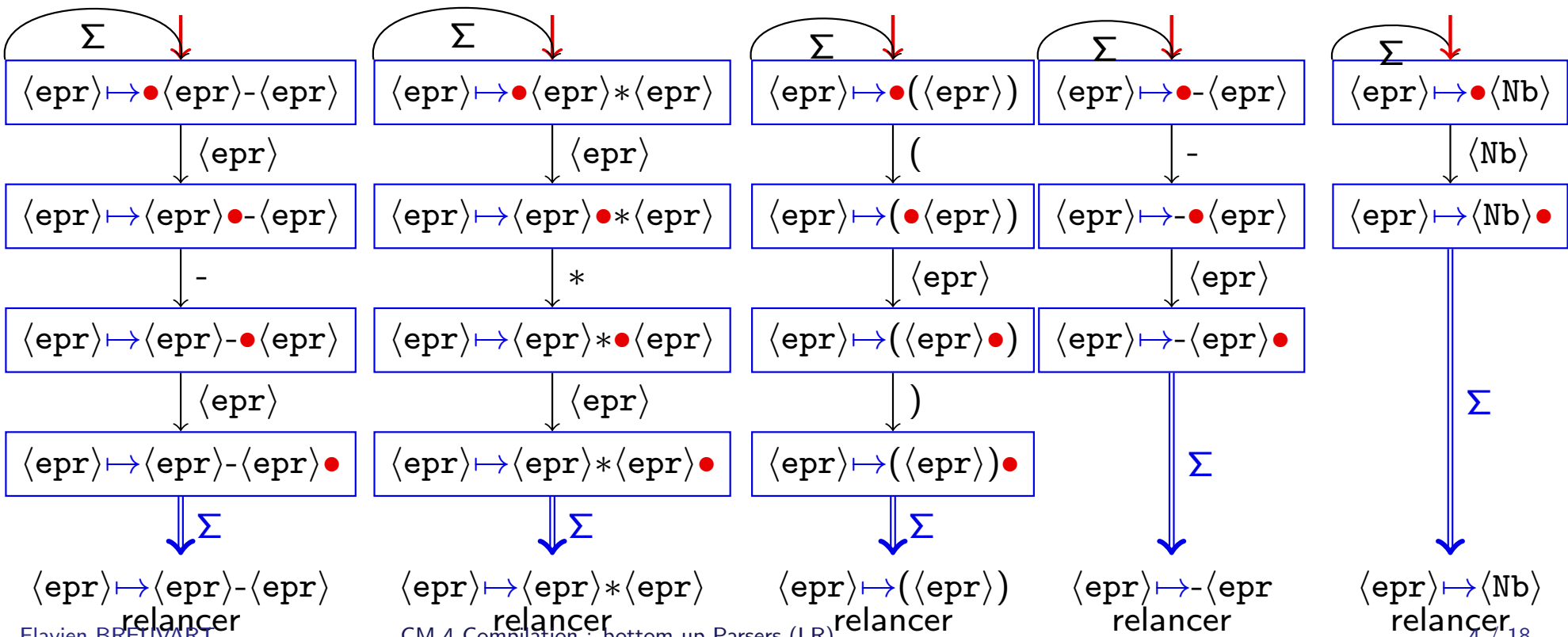
⟨epr⟩↦⟨Nb⟩
relancer

First attempt

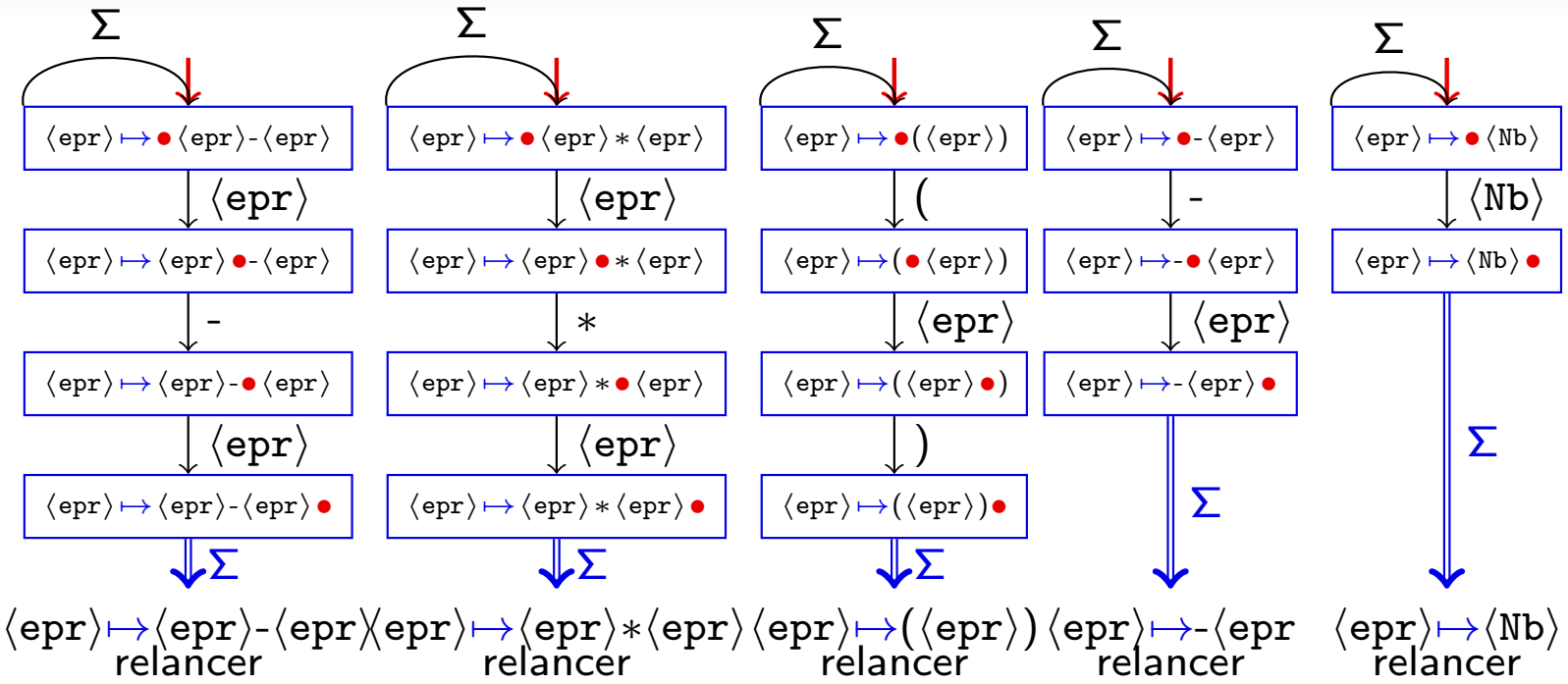
Idea : Same as for the lexer

- a small automaton for each rule that "test" the patern,
- preceded by a delaying loop,
- adds actions and takes the sum.

Remark : such an action is called reduct

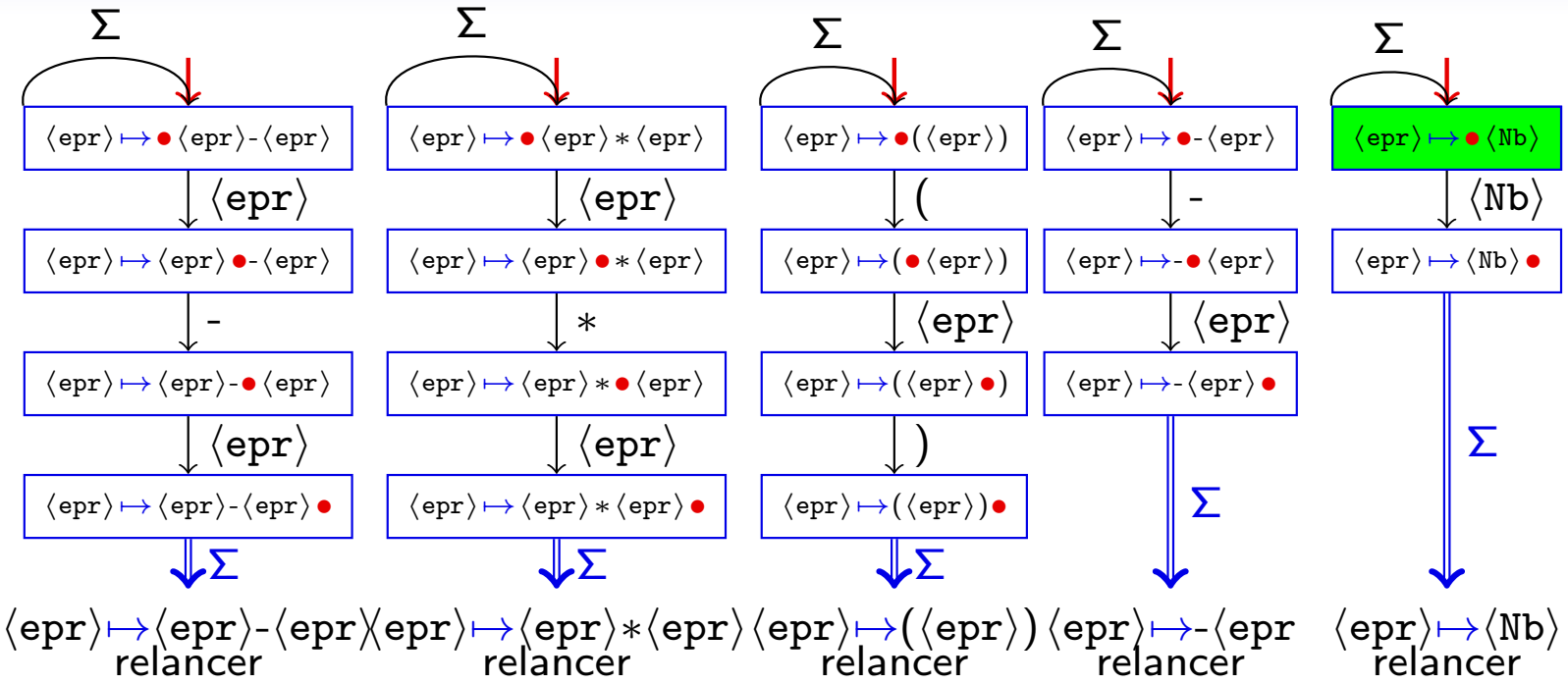


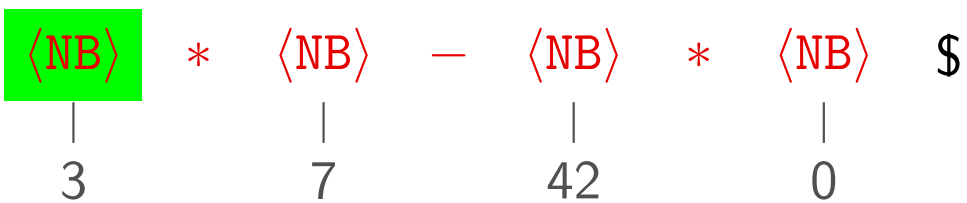
Automaton of the LR parser in motion



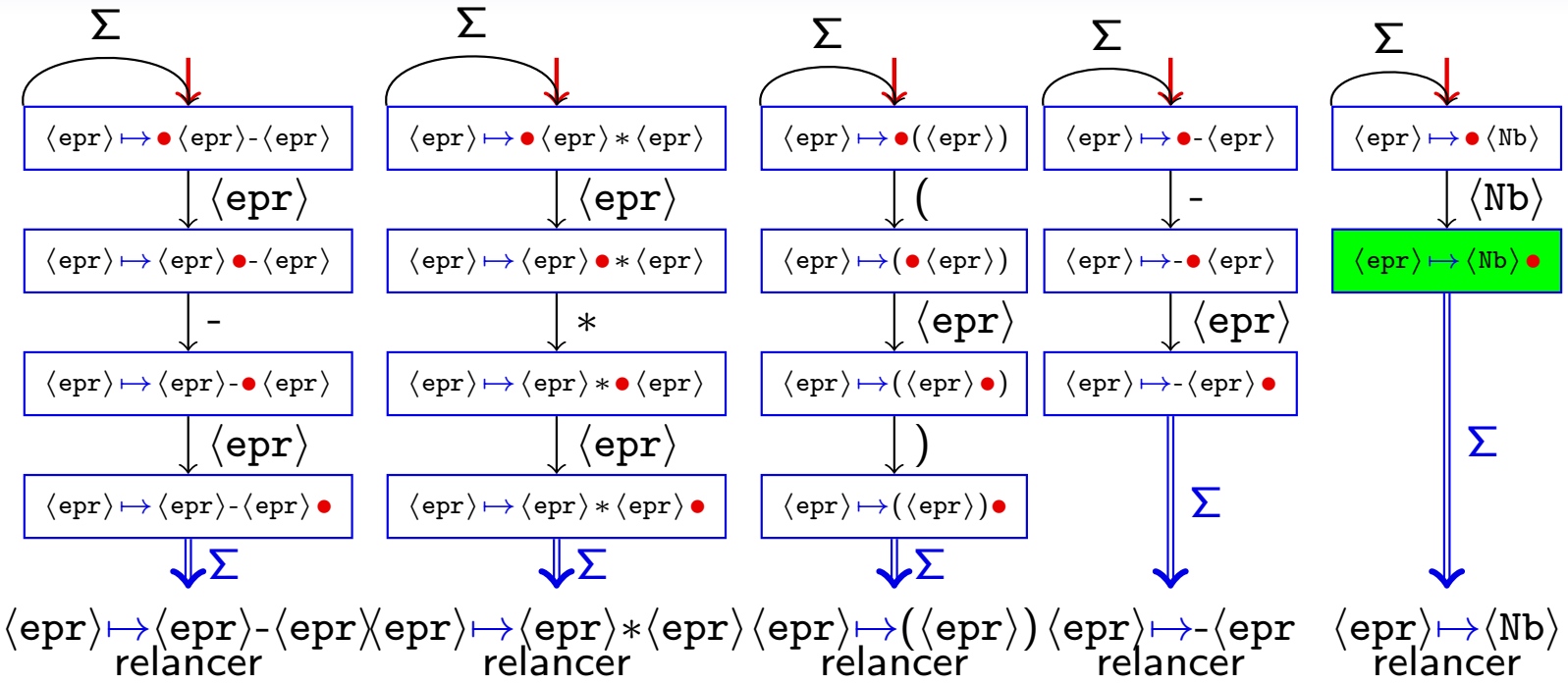
⟨NB⟩	*	⟨NB⟩	−	⟨NB⟩	*	⟨NB⟩	$
3		7		42		0	

Automaton of the LR parser in motion



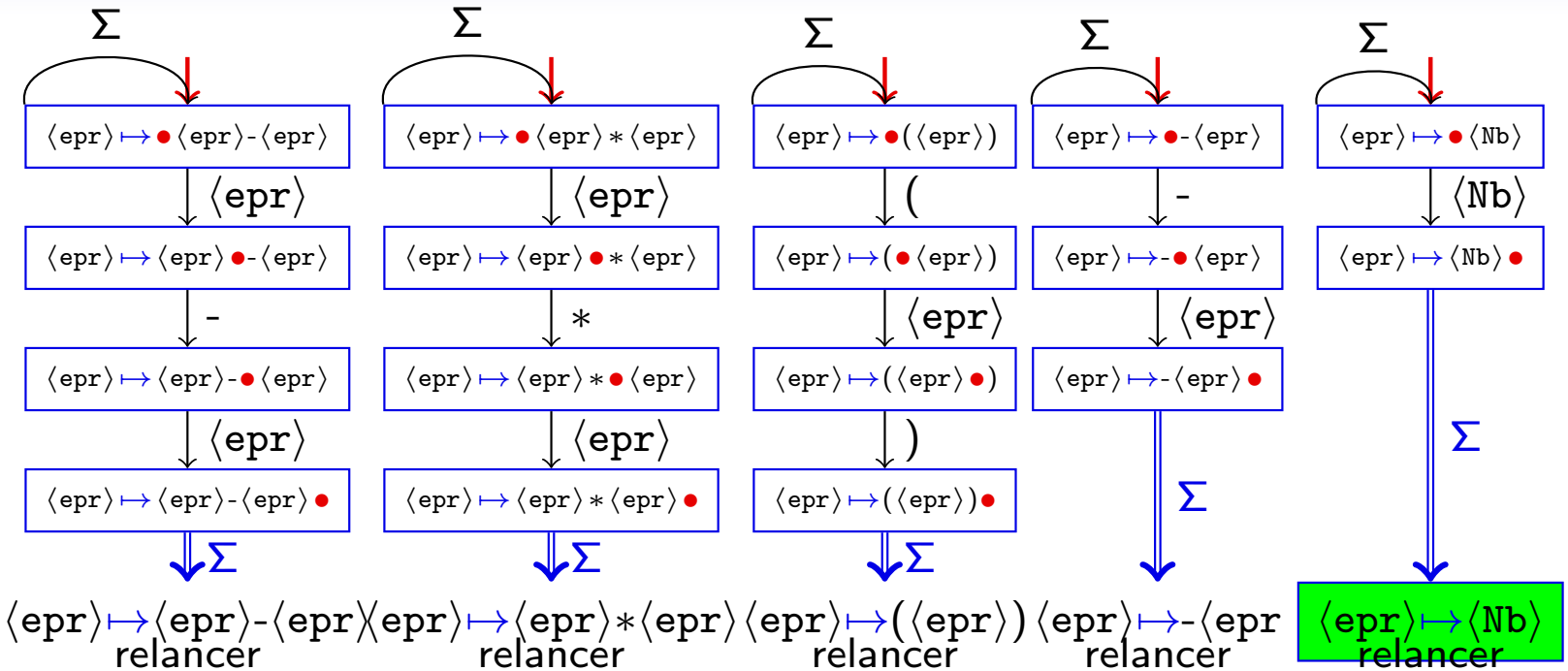


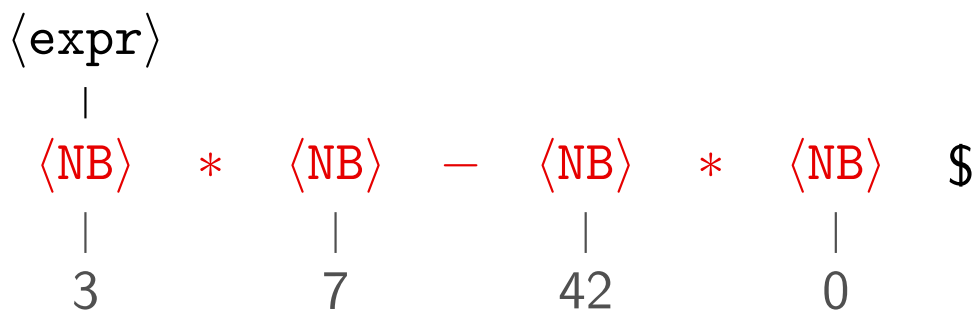
Automaton of the LR parser in motion



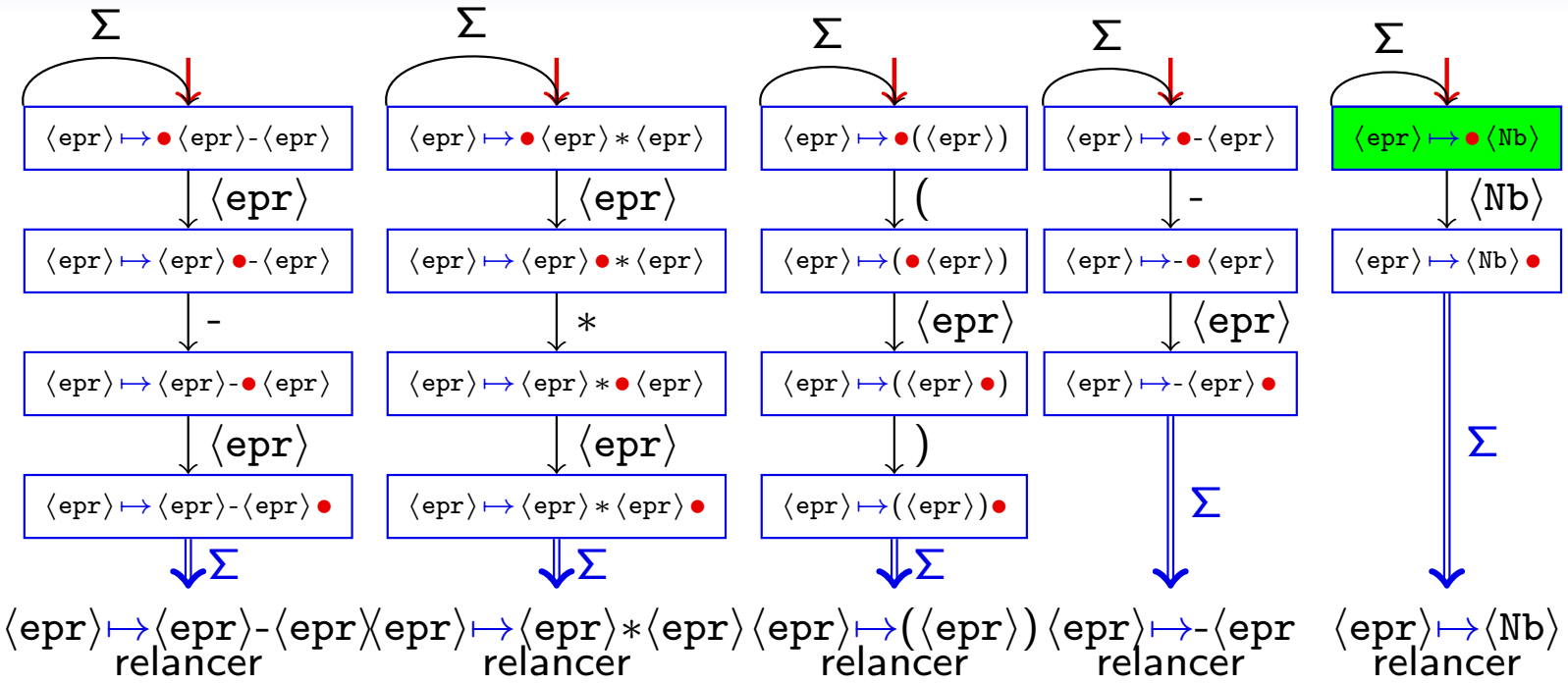
⟨NB⟩	*	⟨NB⟩	−	⟨NB⟩	*	⟨NB⟩	$
|		|		|		|	
3		7		42		0	

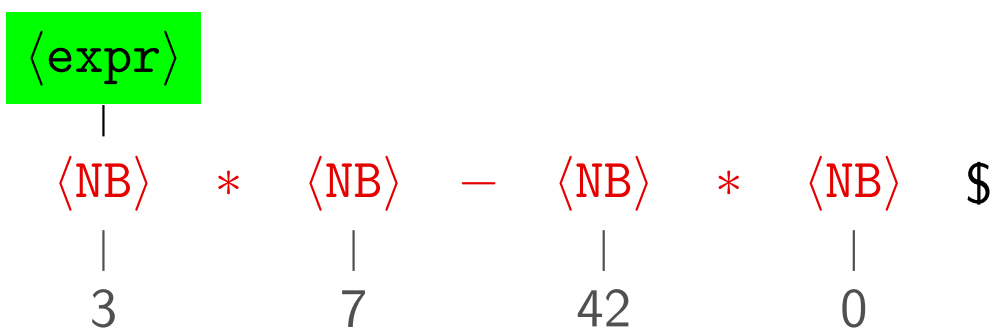
Automaton of the LR parser in motion



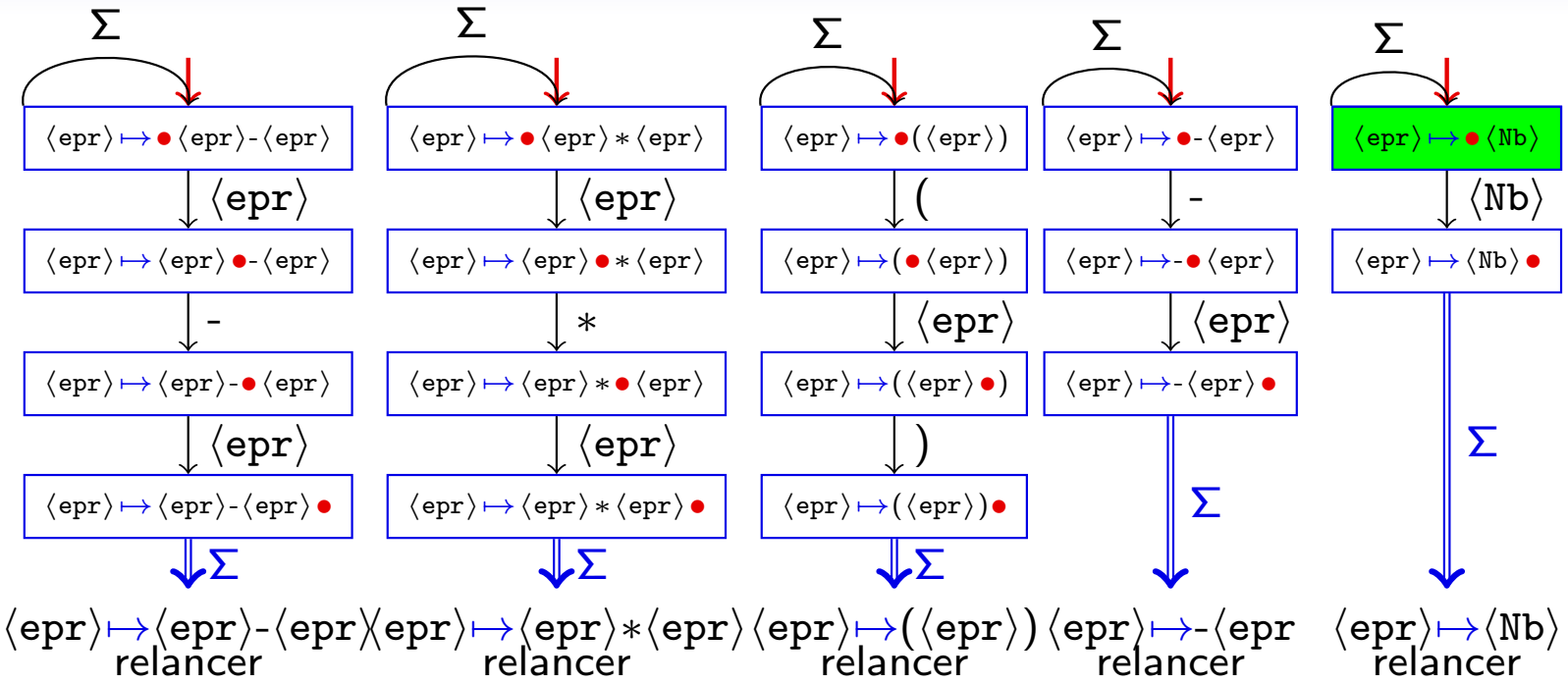


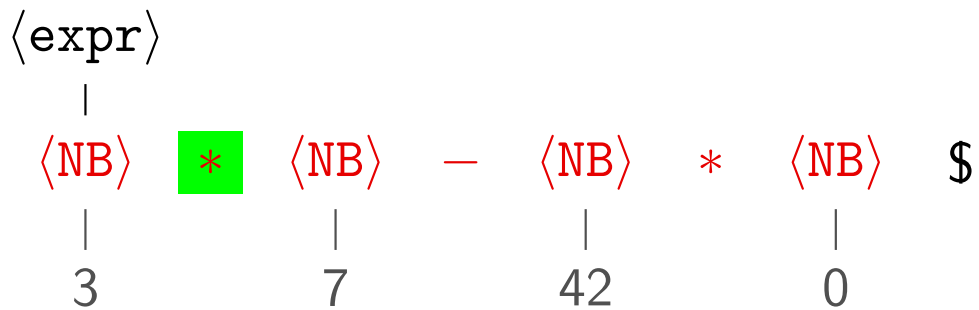
Automaton of the LR parser in motion



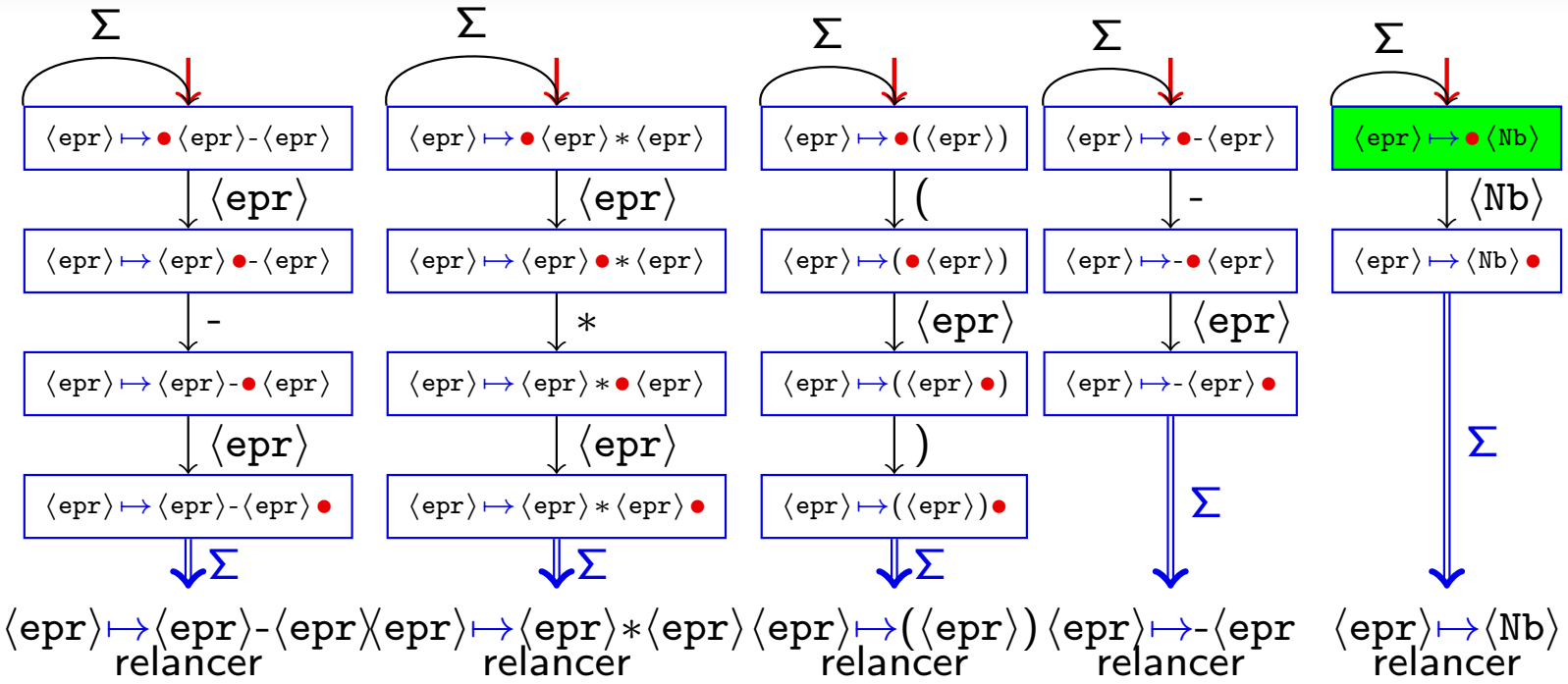


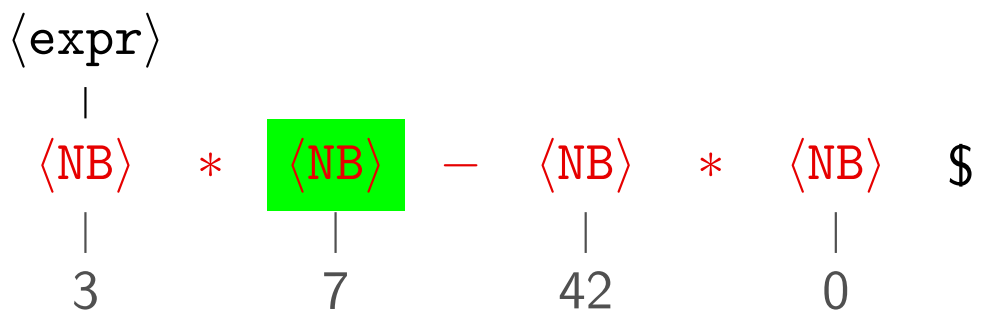
Automaton of the LR parser in motion



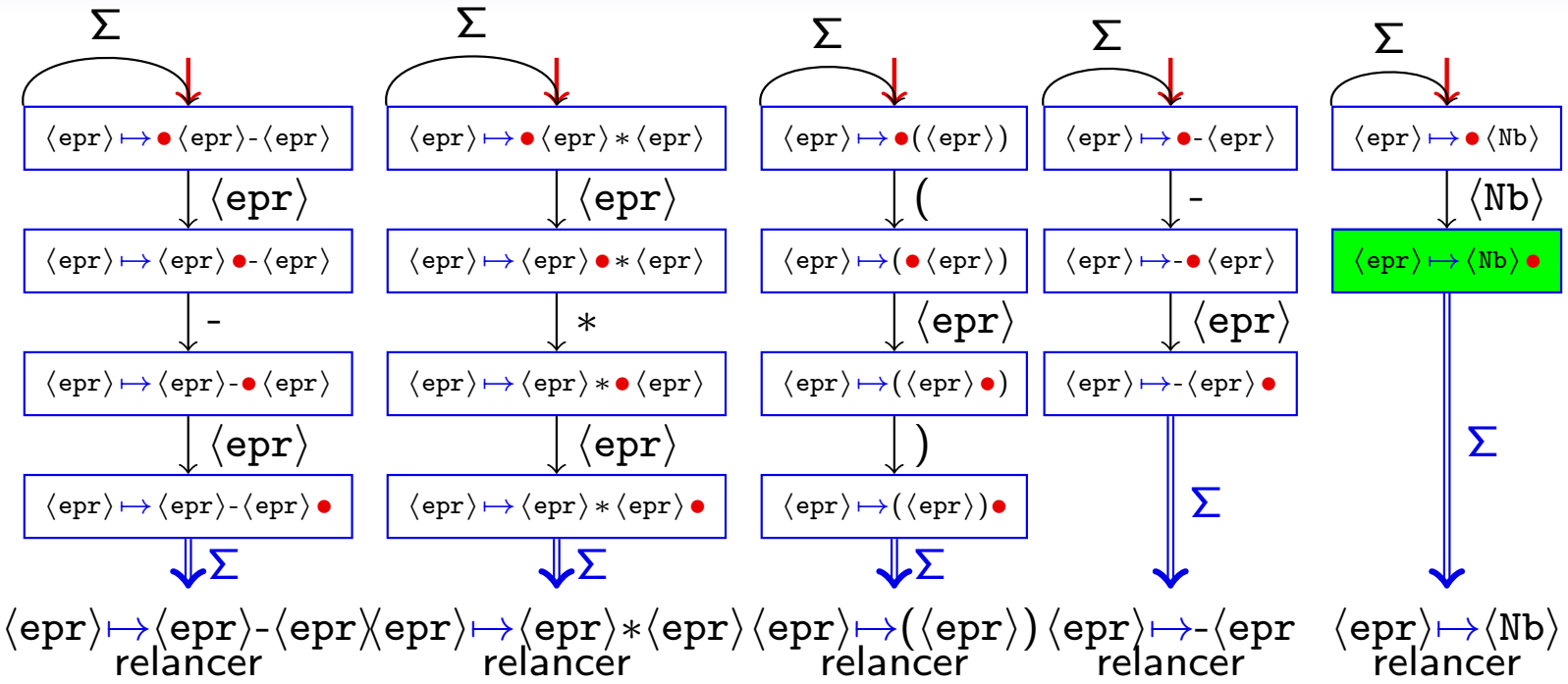


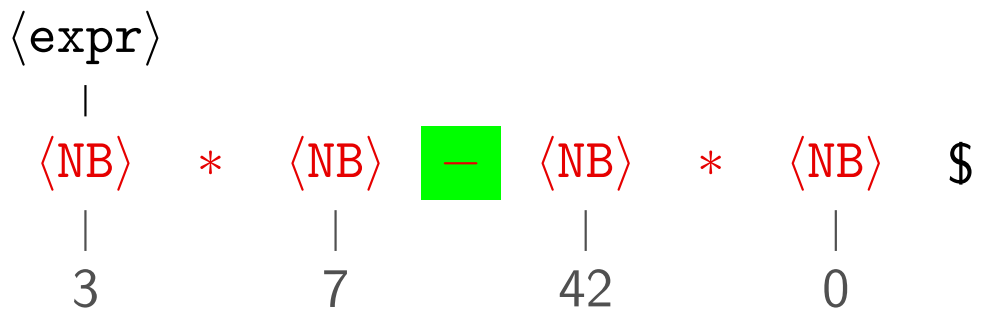
Automaton of the LR parser in motion



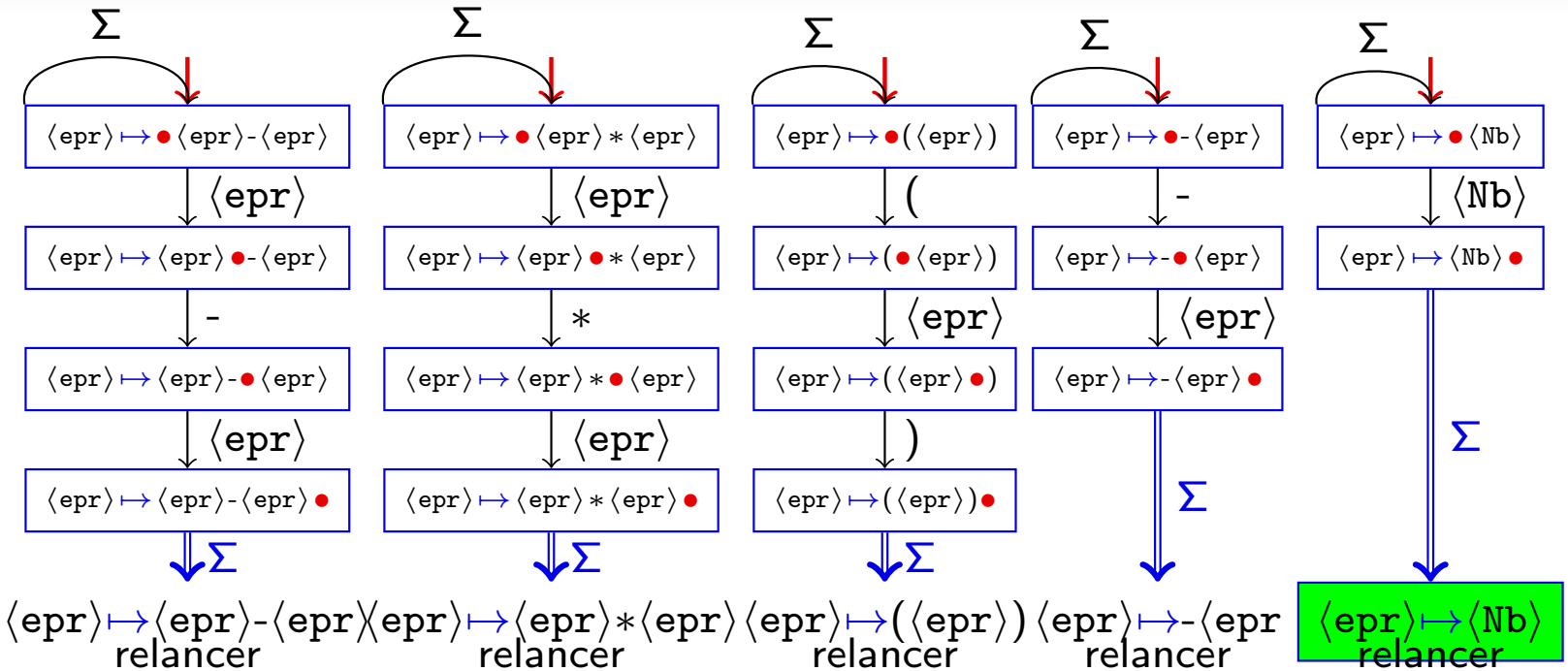


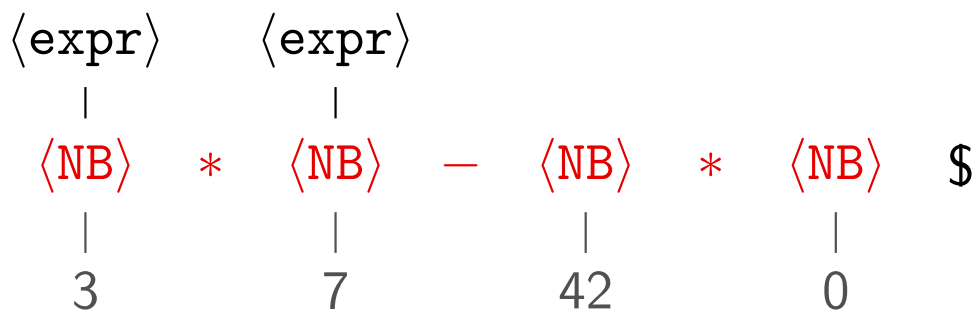
Automaton of the LR parser in motion



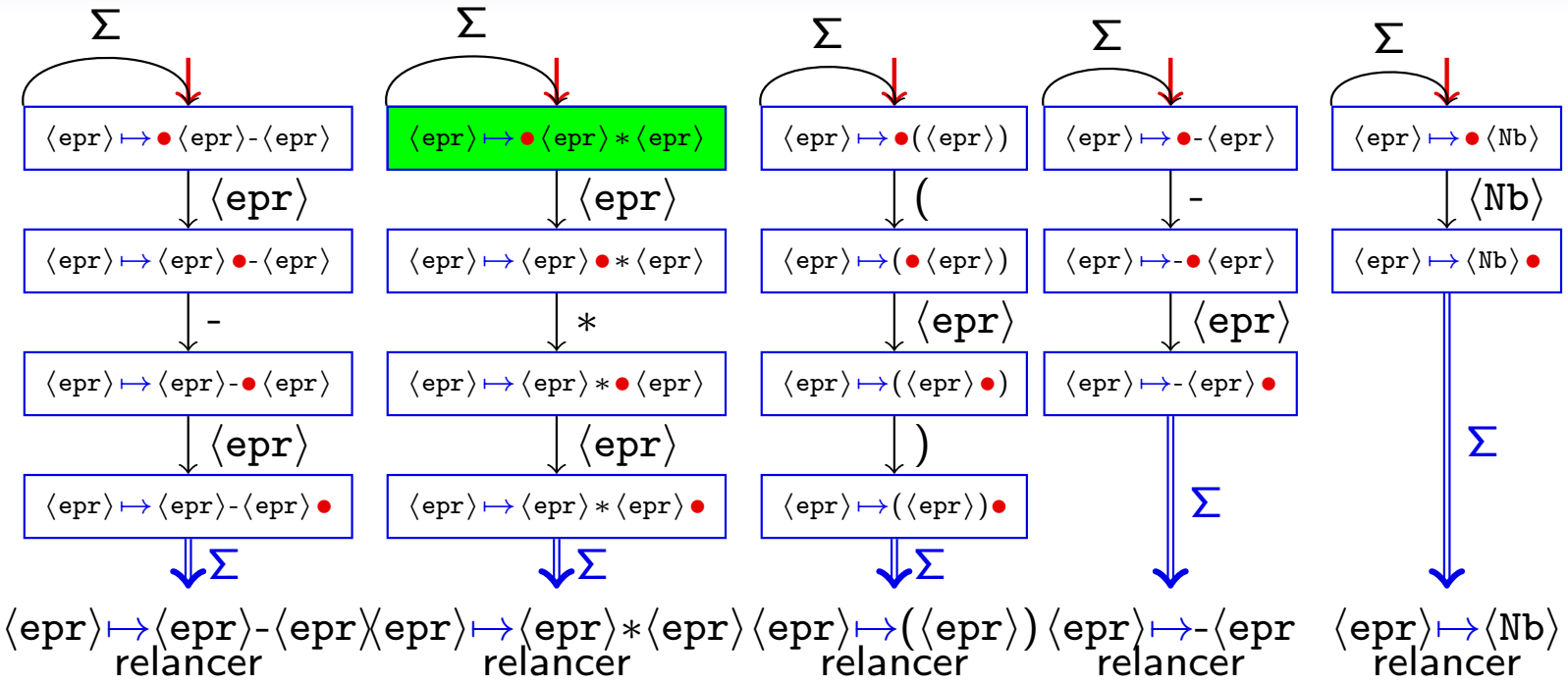


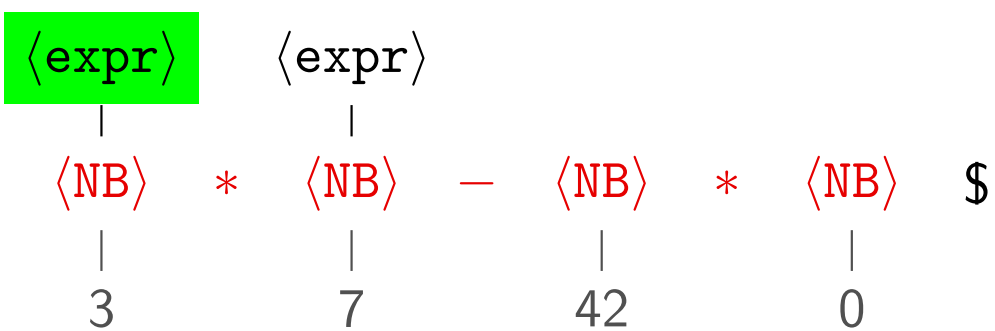
Automaton of the LR parser in motion



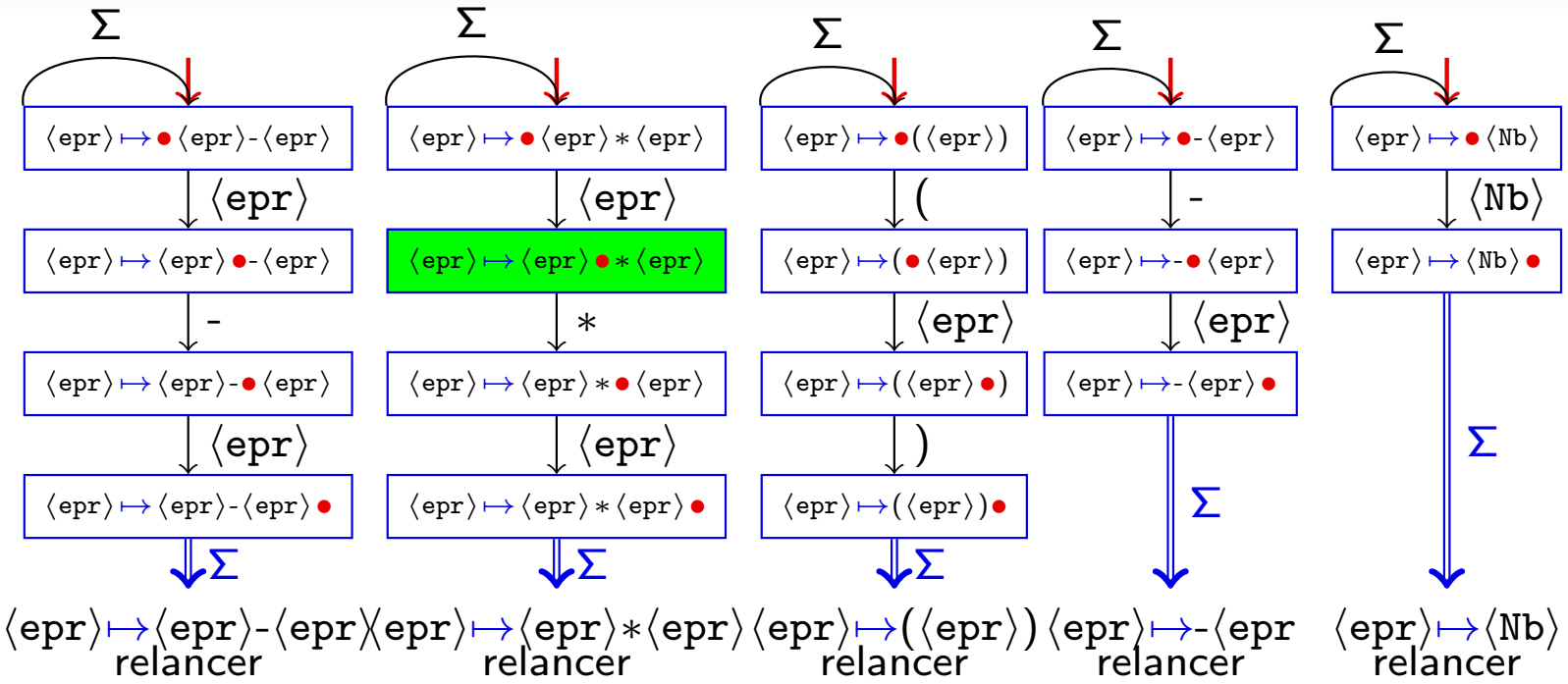


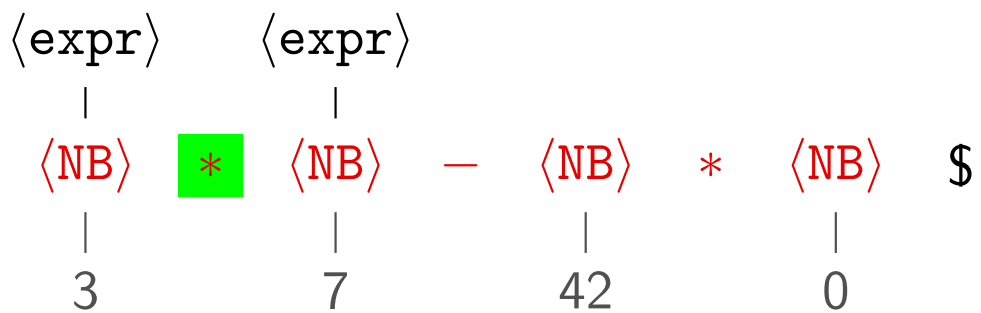
Automaton of the LR parser in motion



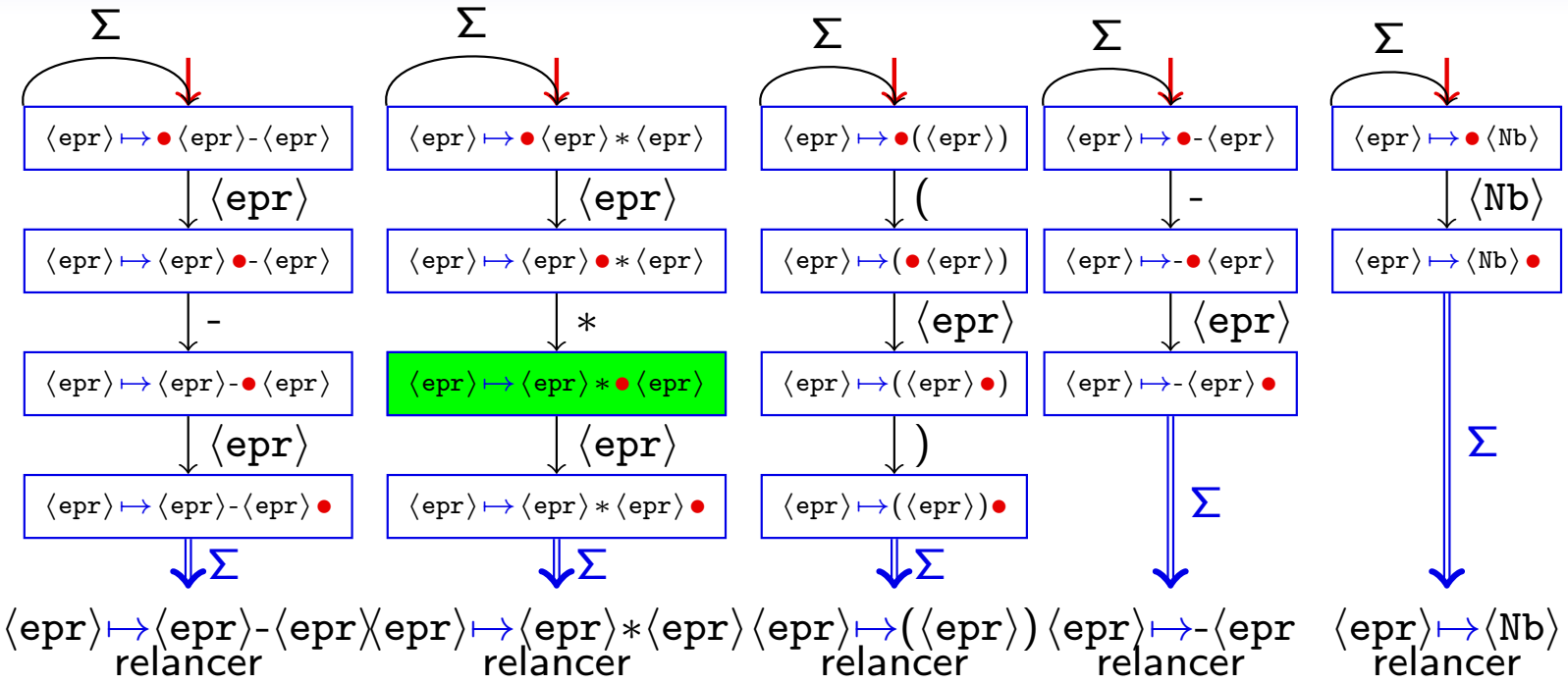


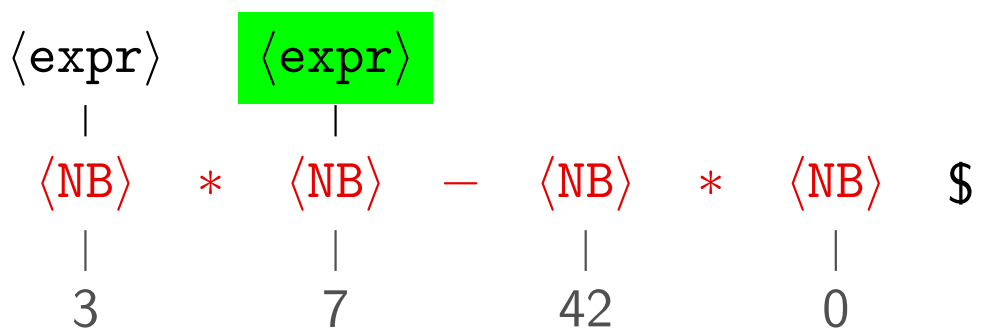
Automaton of the LR parser in motion



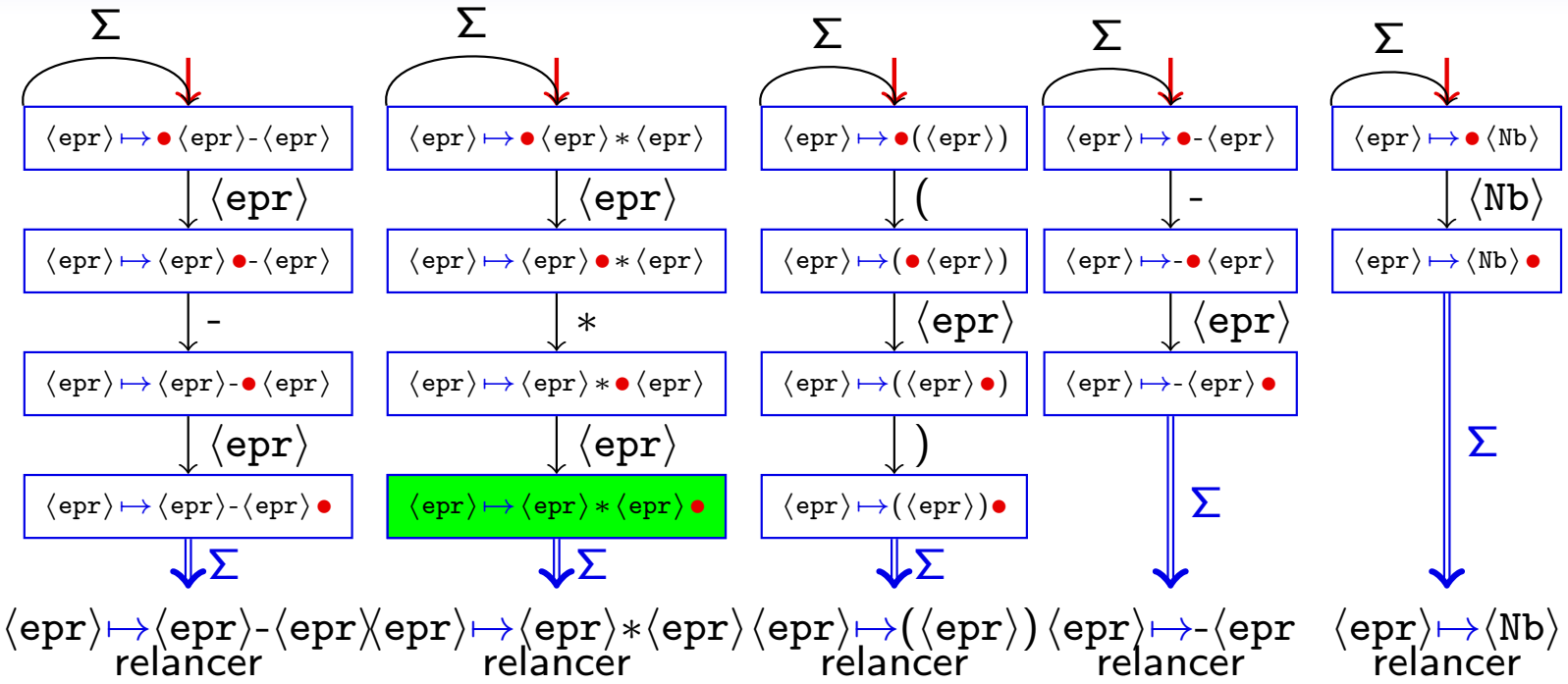


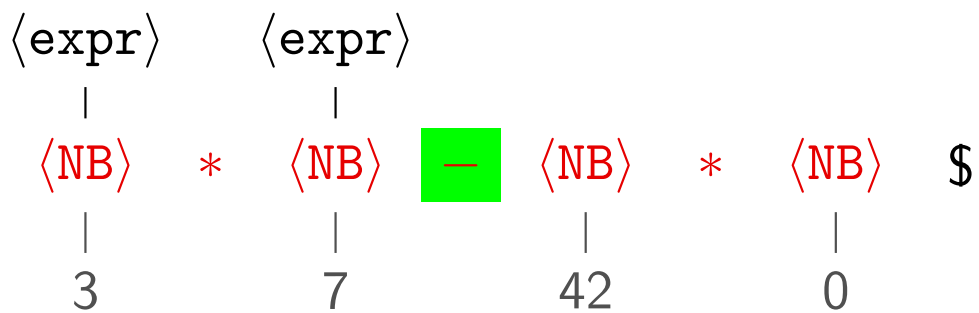
Automaton of the LR parser in motion



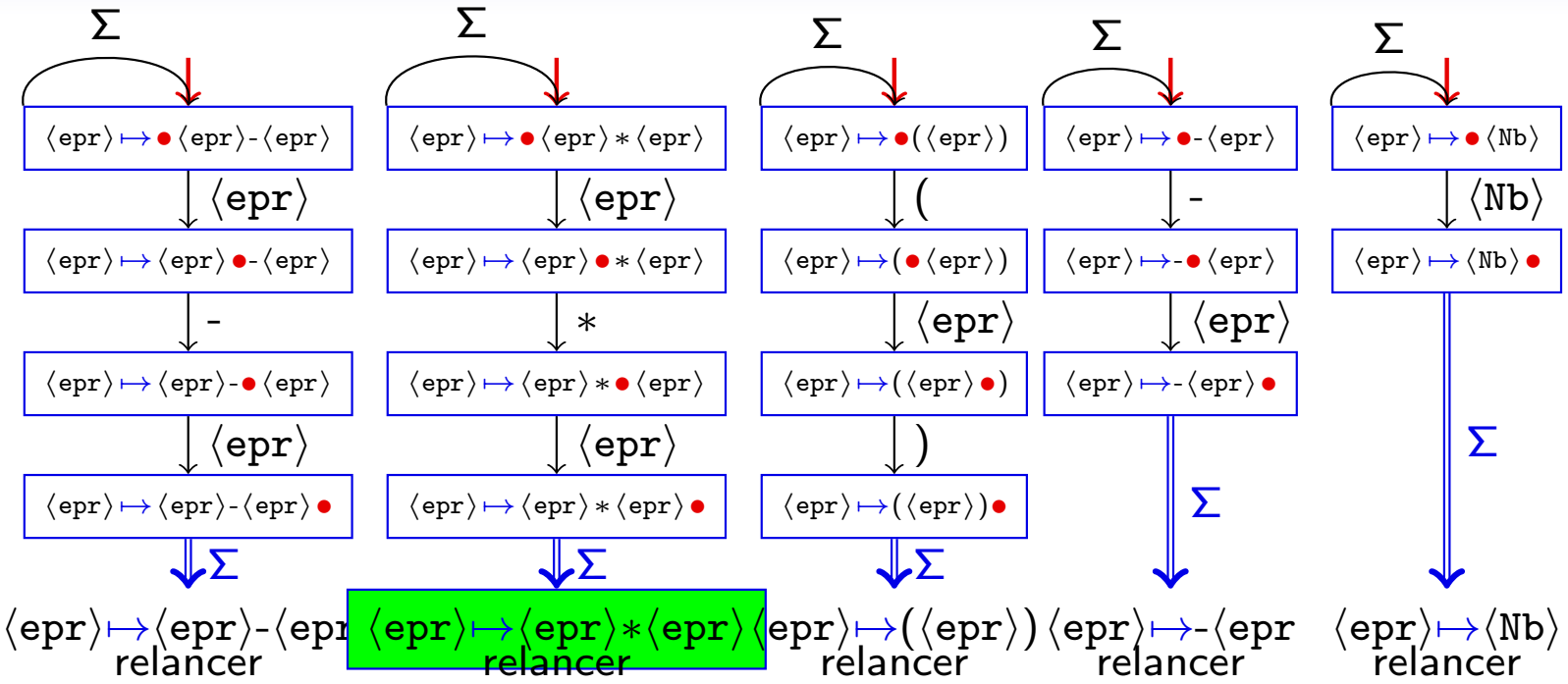


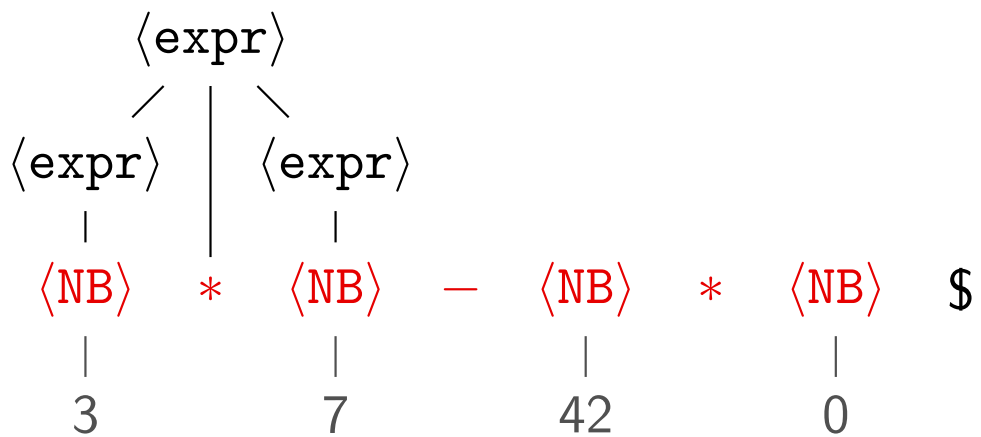
Automaton of the LR parser in motion



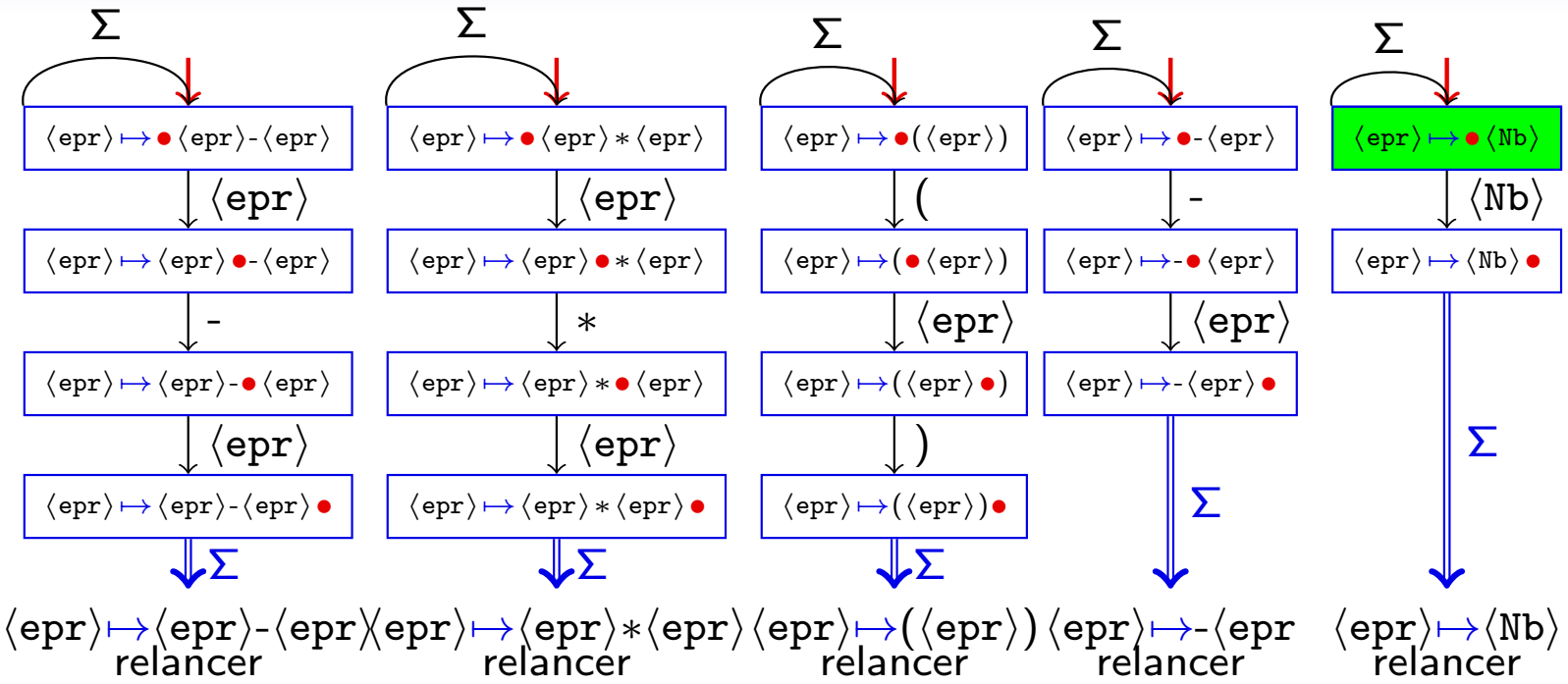


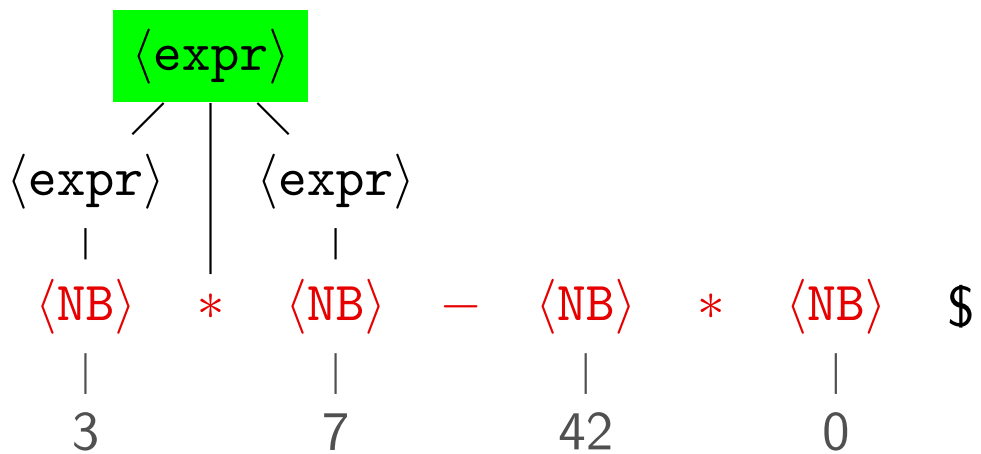
Automaton of the LR parser in motion



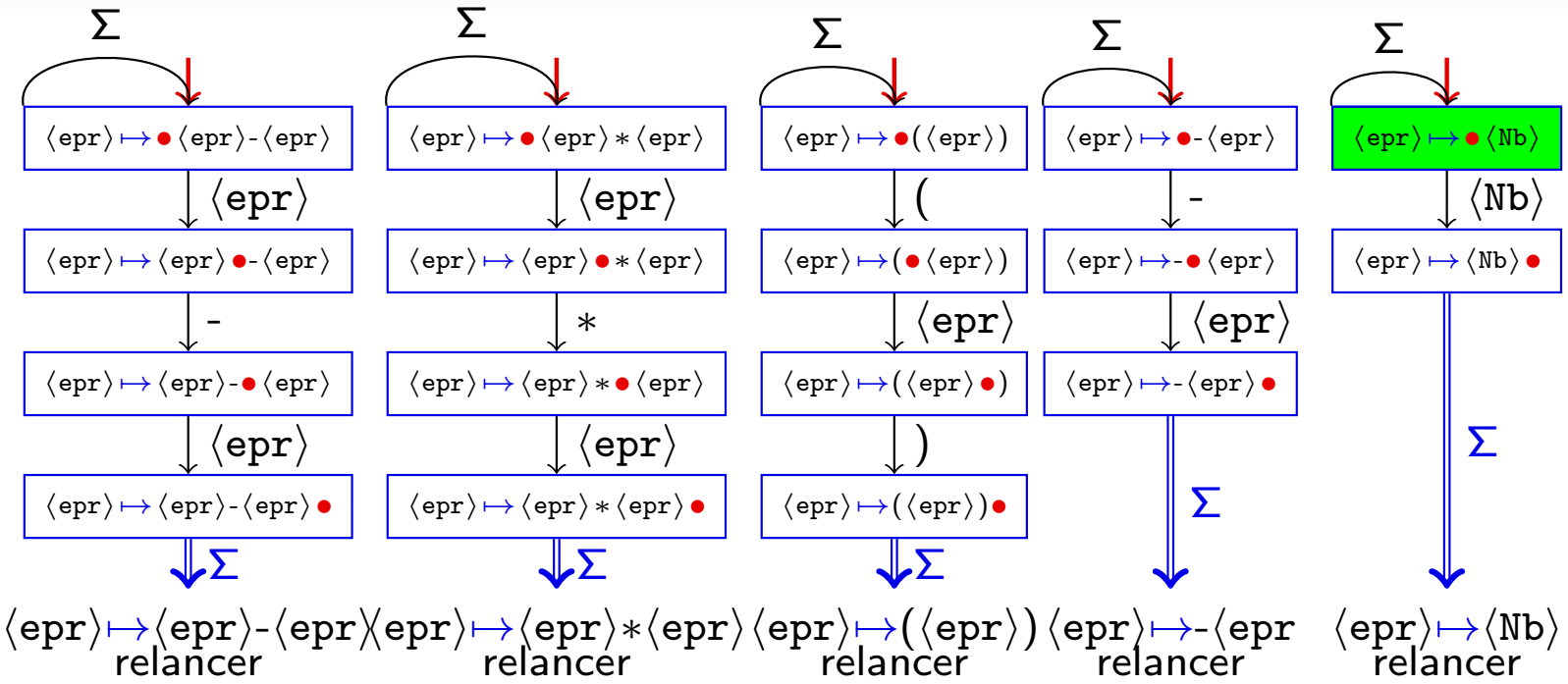


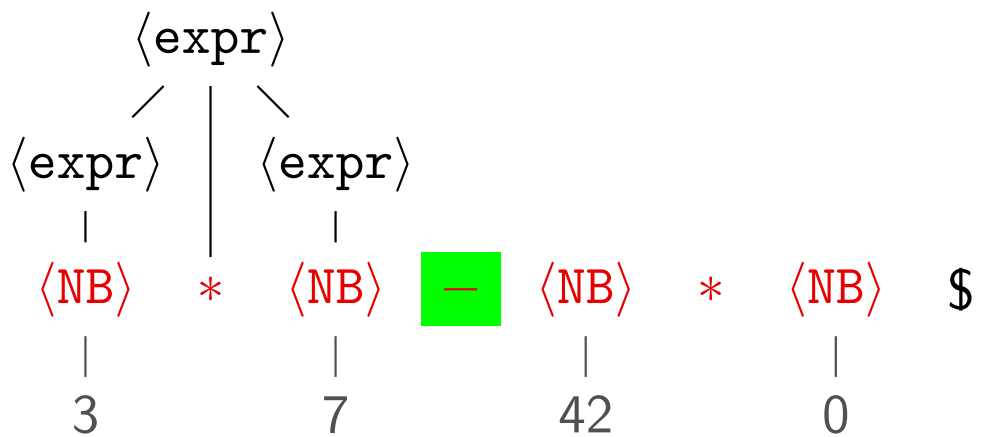
Automaton of the LR parser in motion



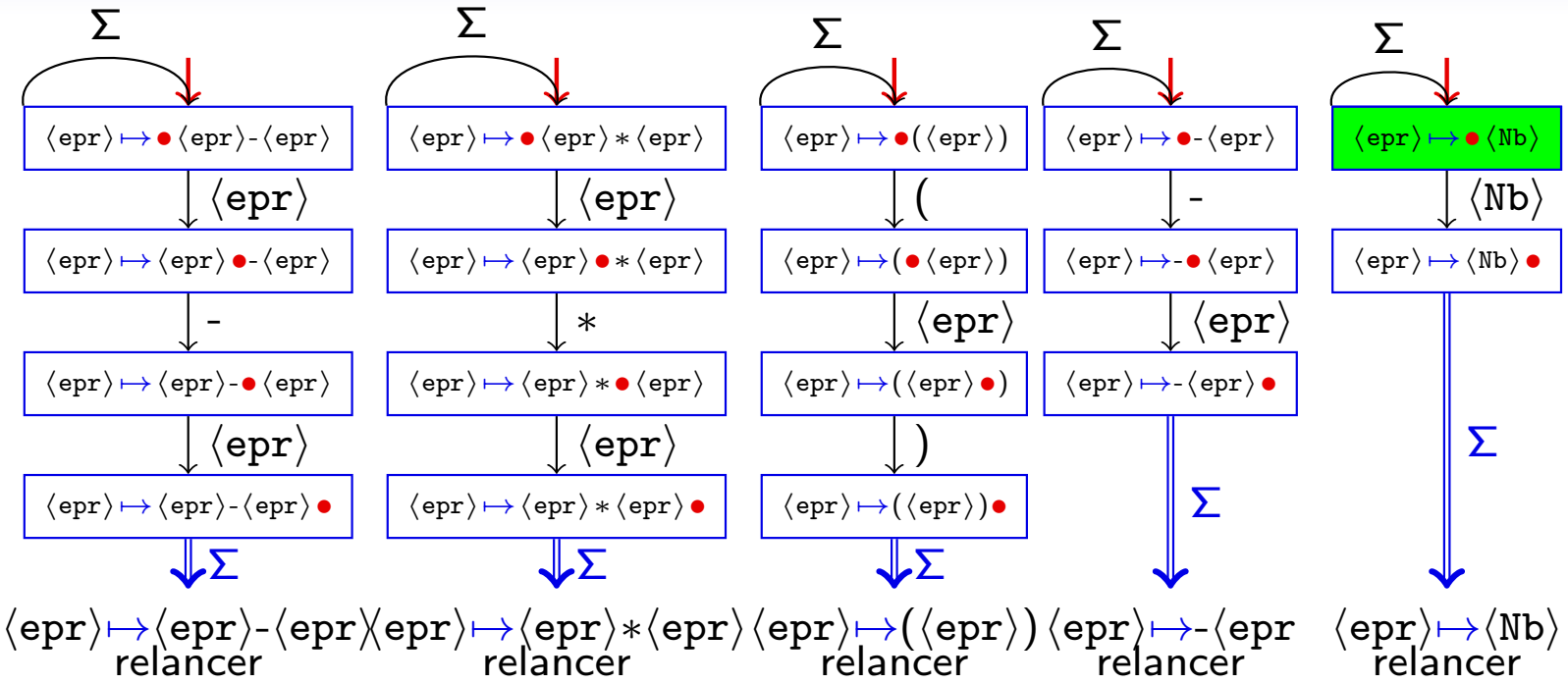


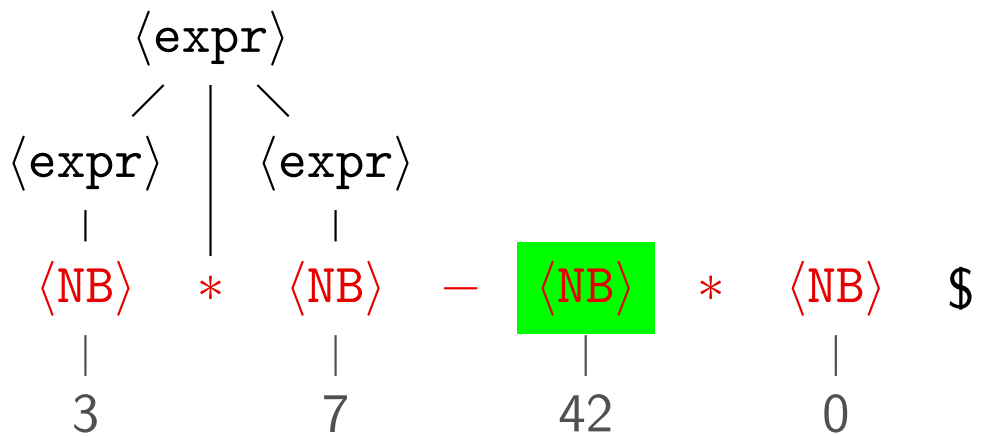
Automaton of the LR parser in motion



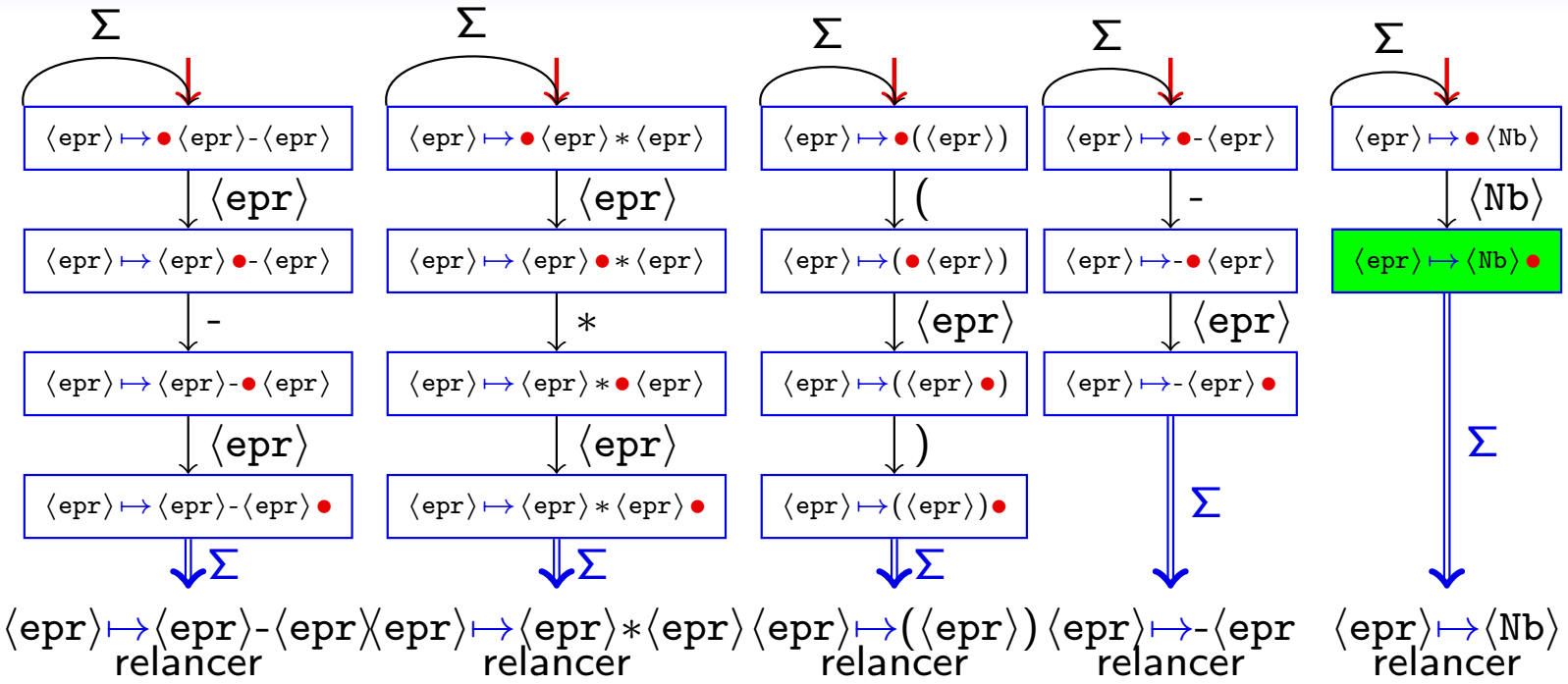


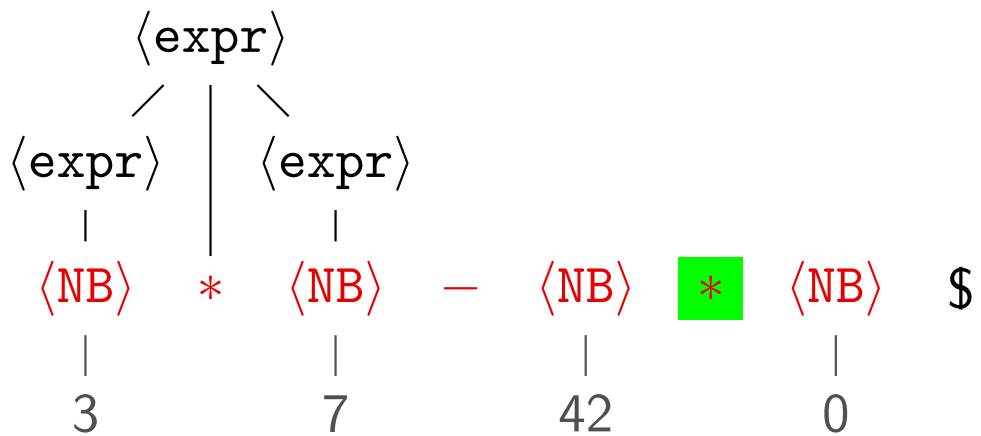
Automaton of the LR parser in motion



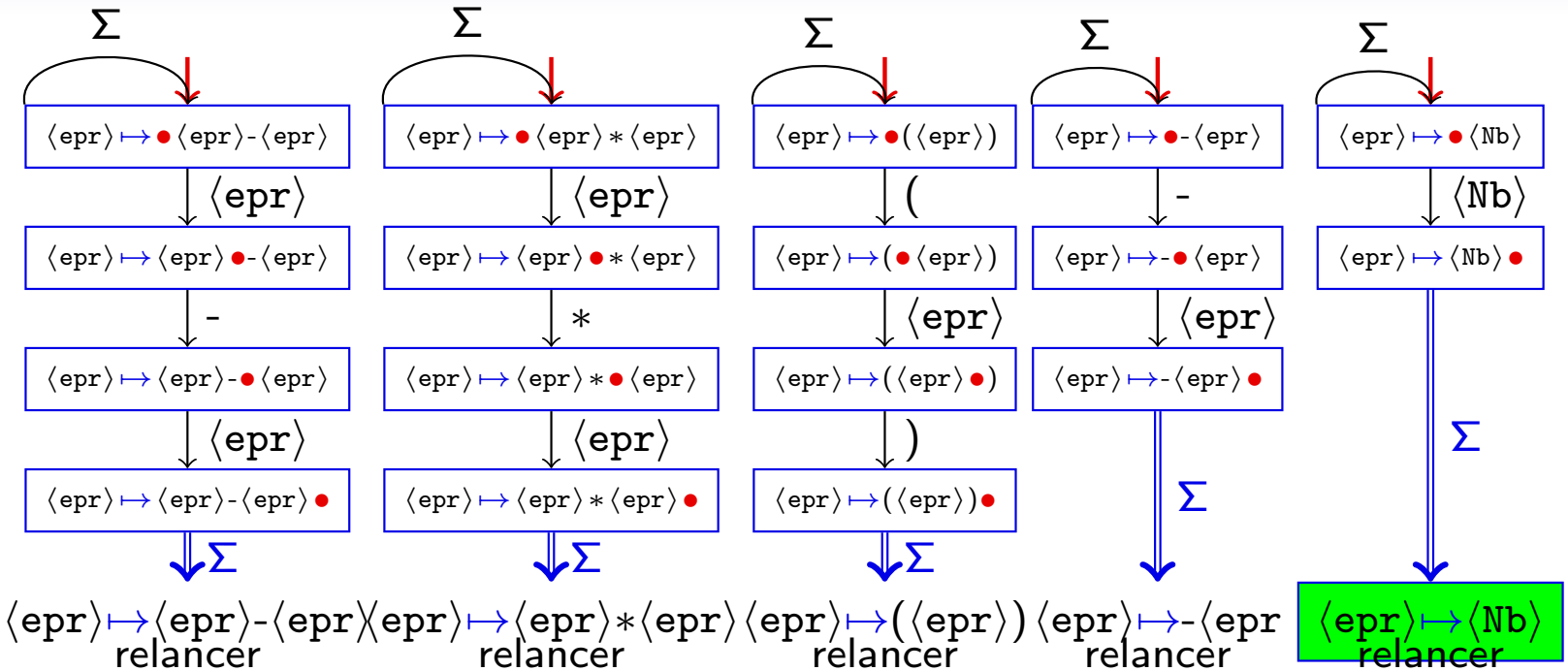


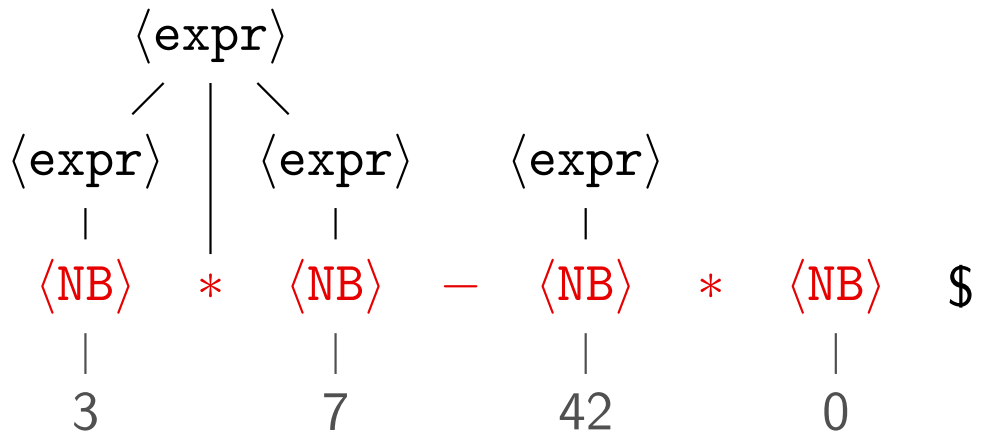
Automaton of the LR parser in motion



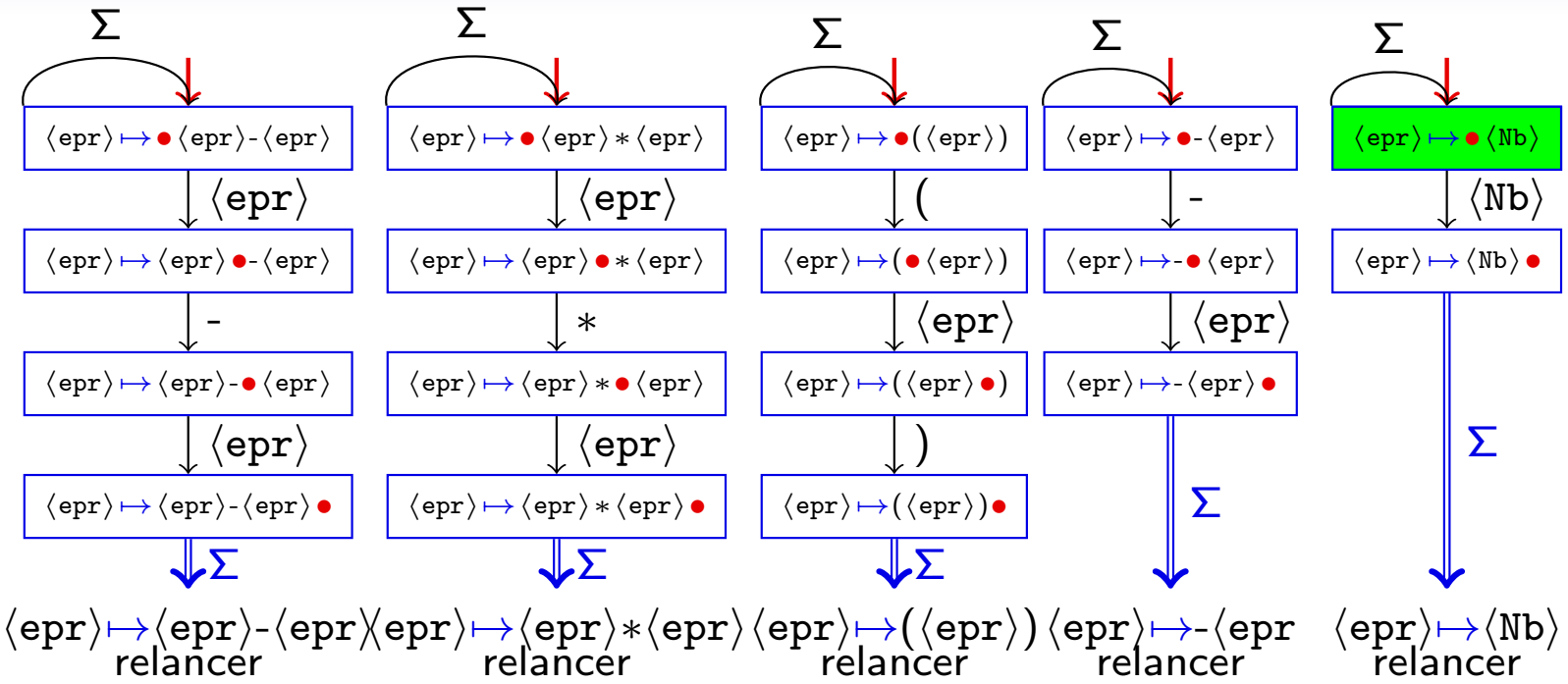


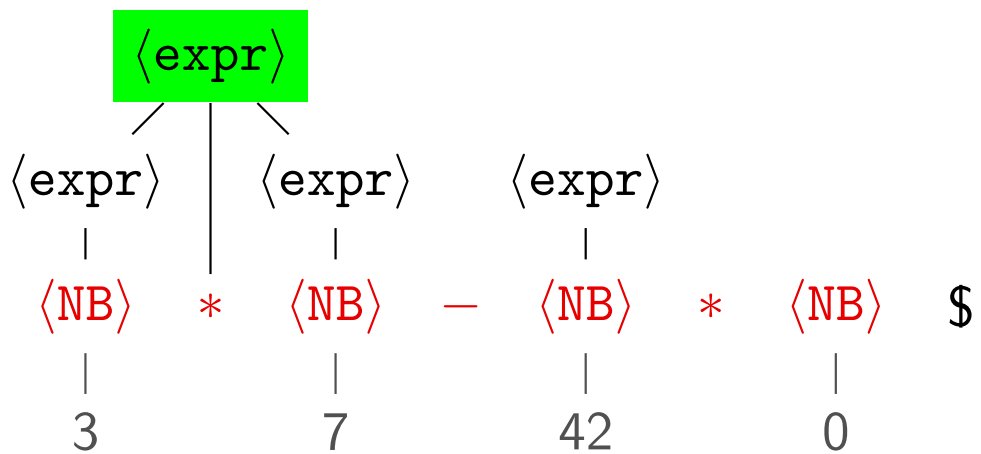
Automaton of the LR parser in motion



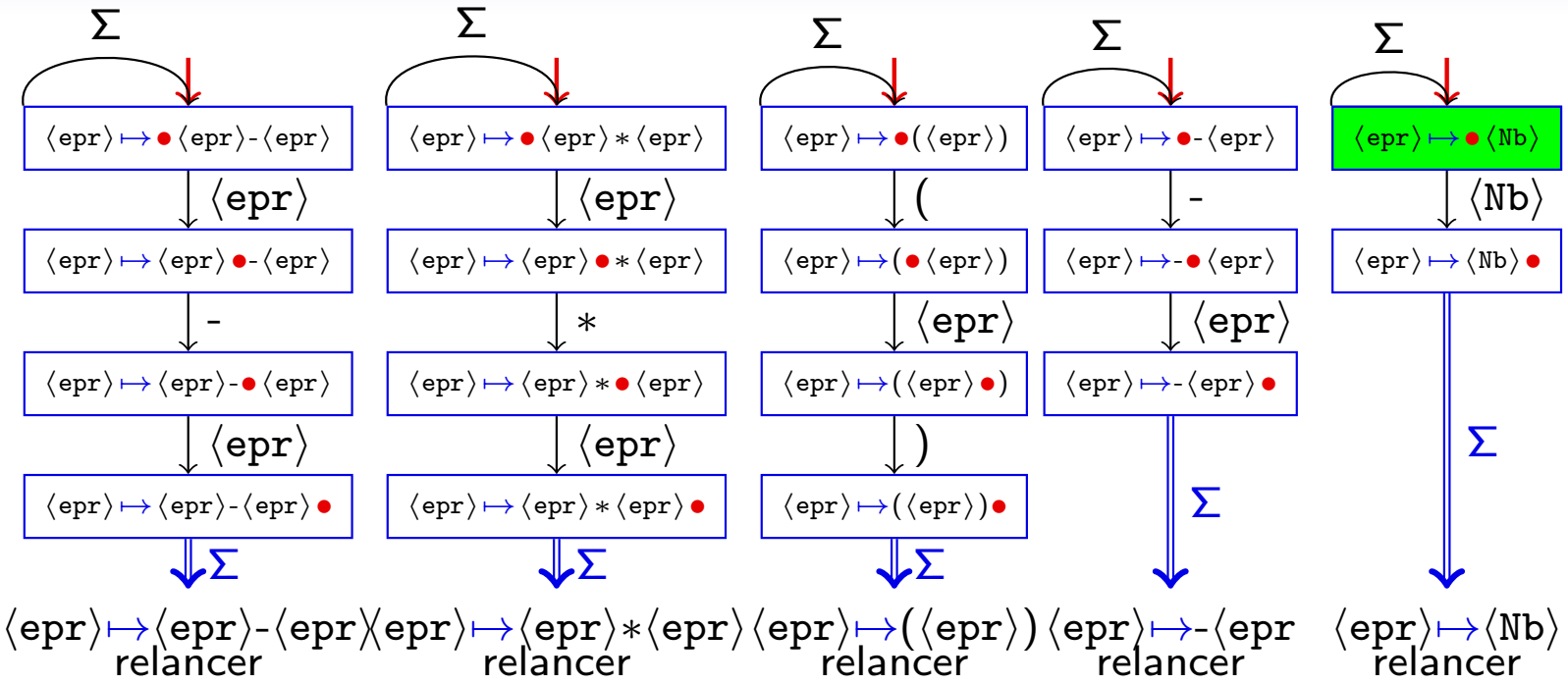


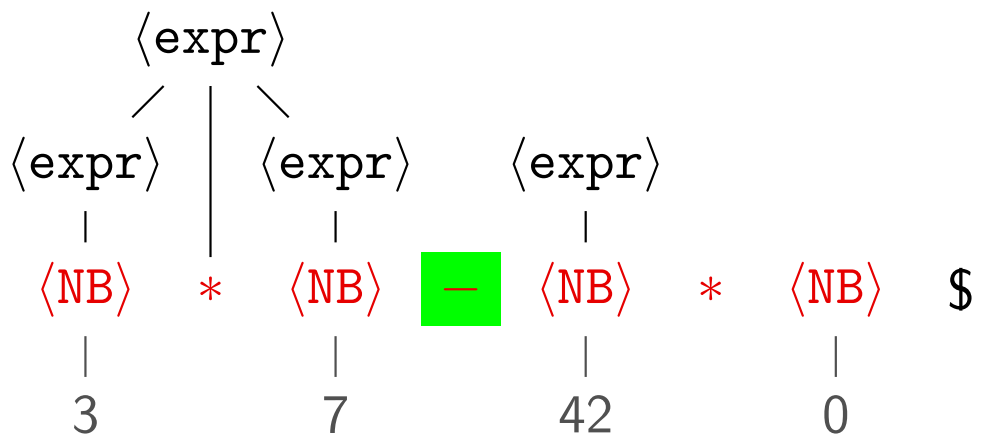
Automaton of the LR parser in motion



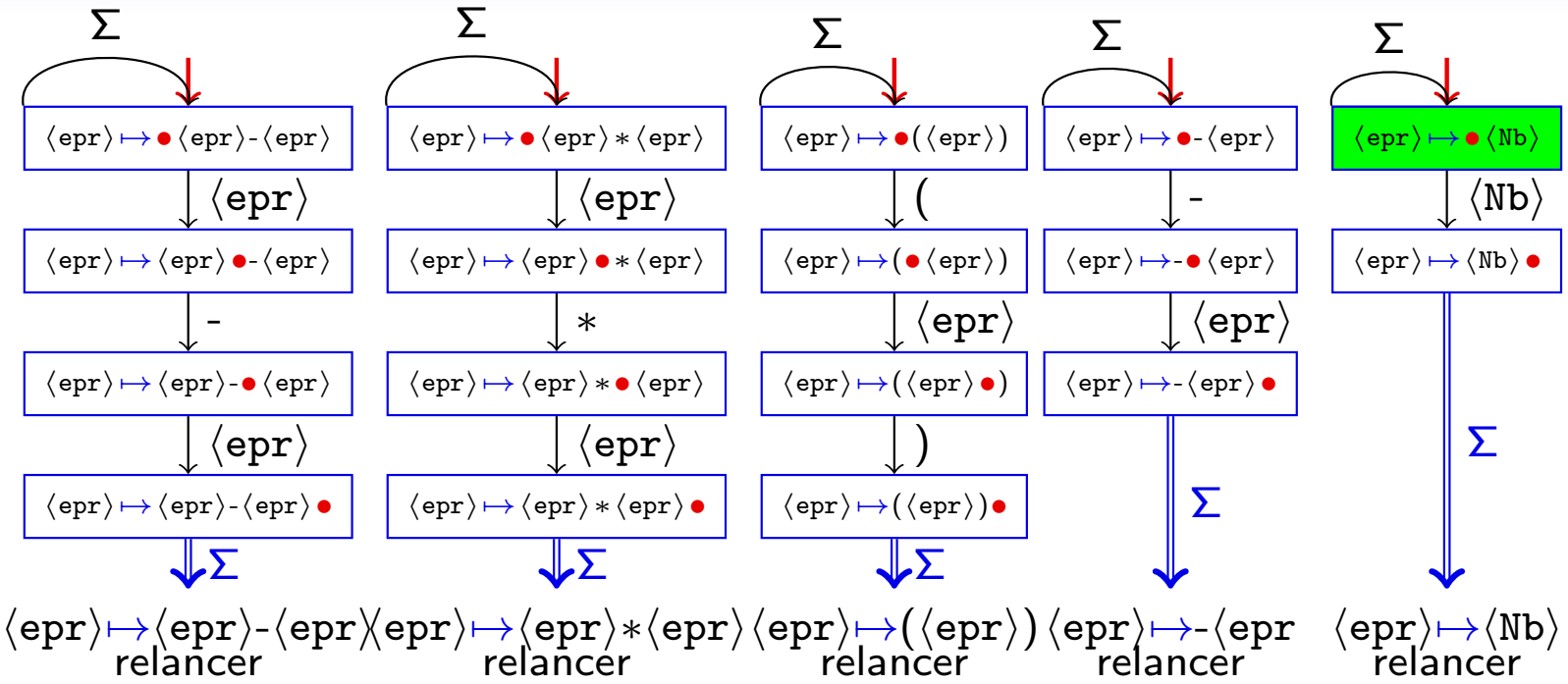


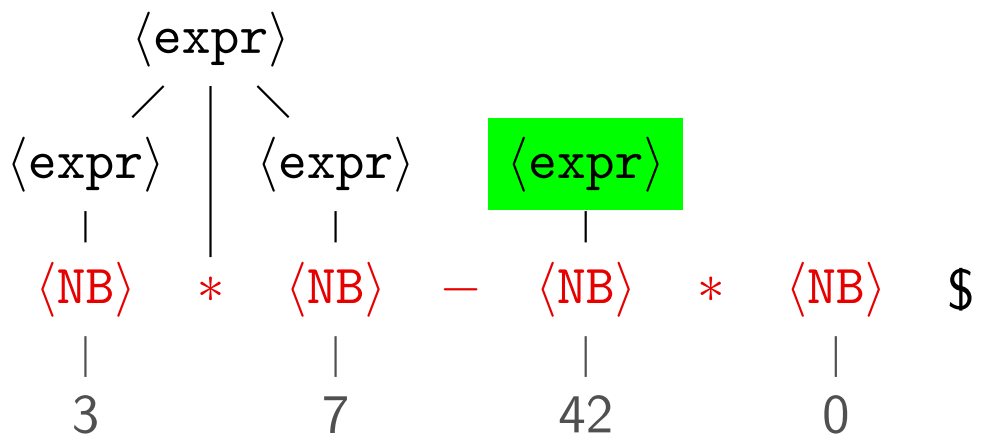
Automaton of the LR parser in motion



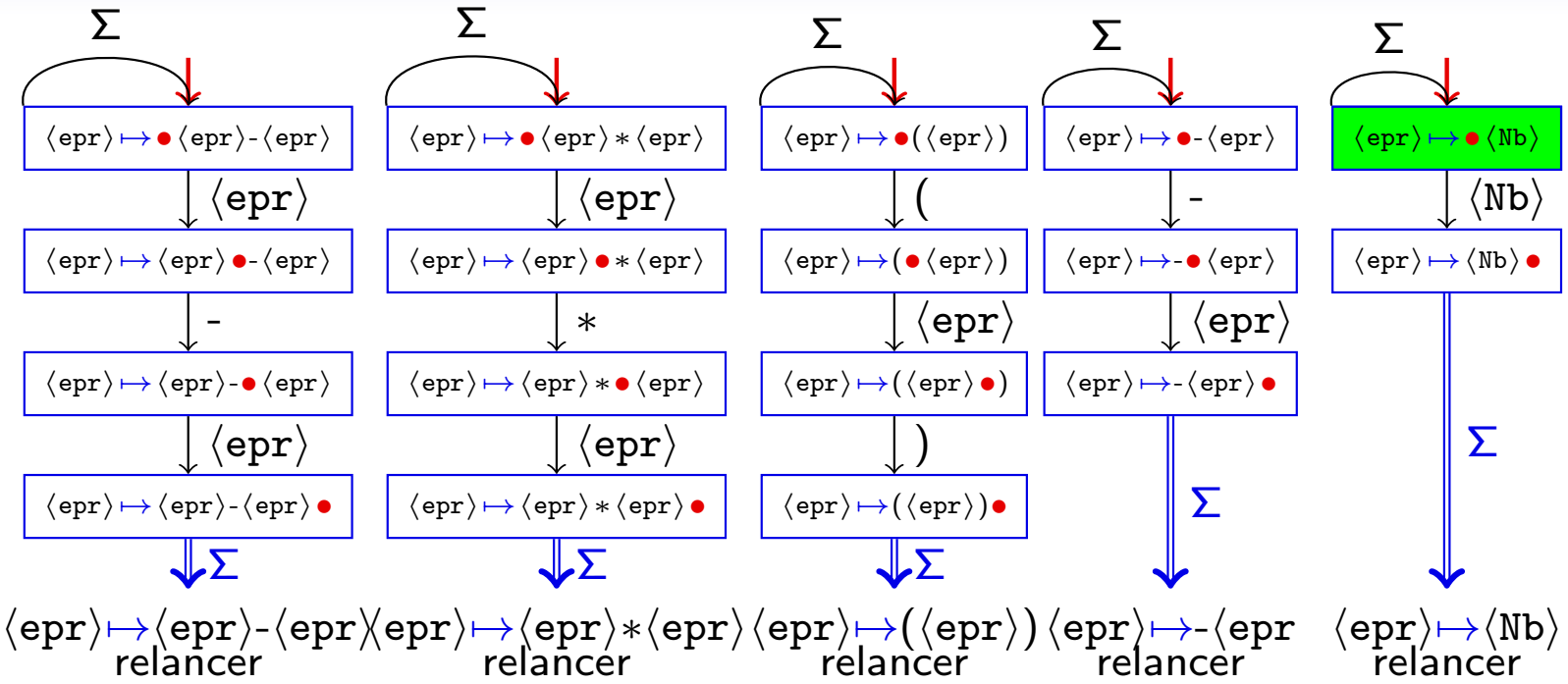


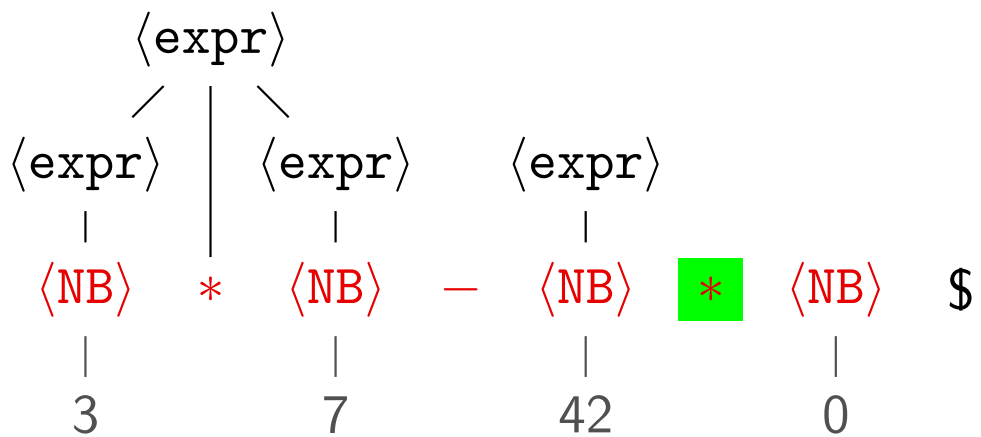
Automaton of the LR parser in motion



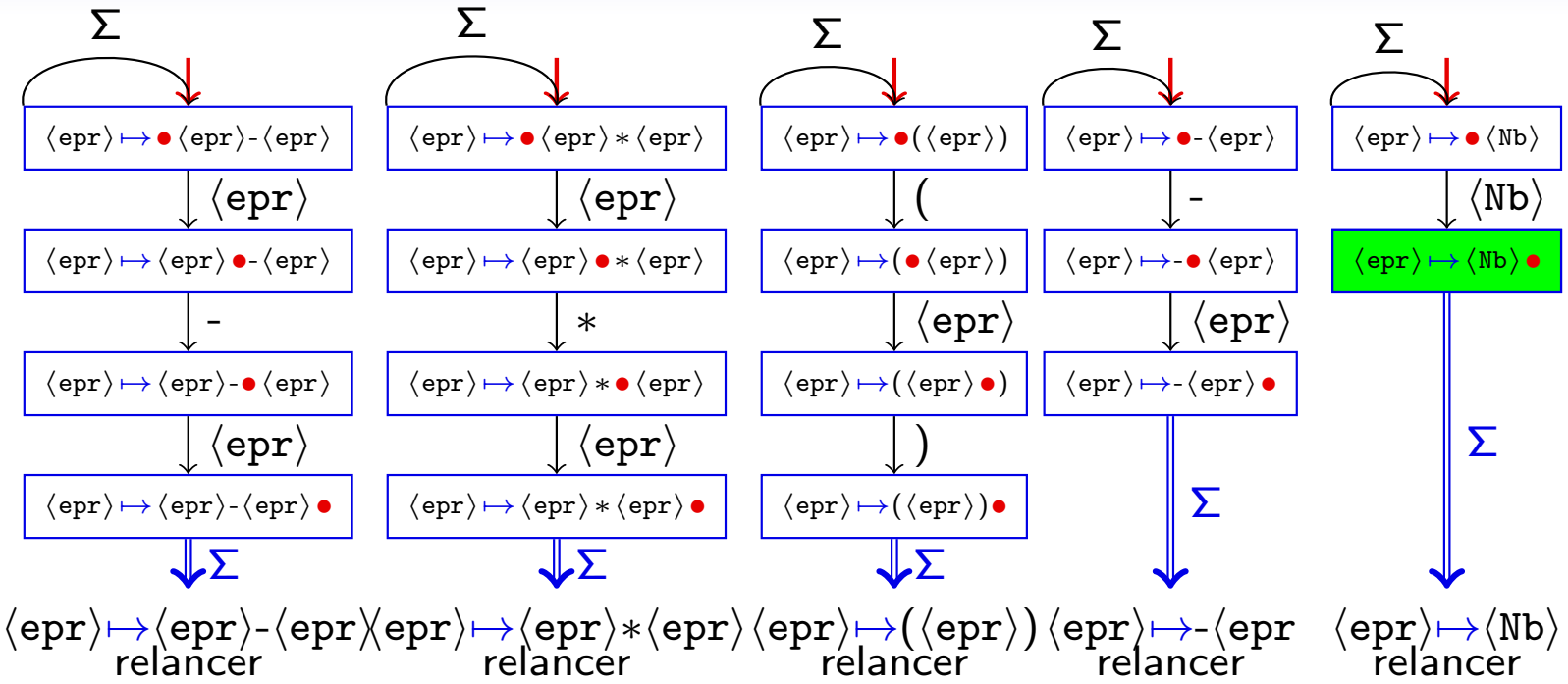


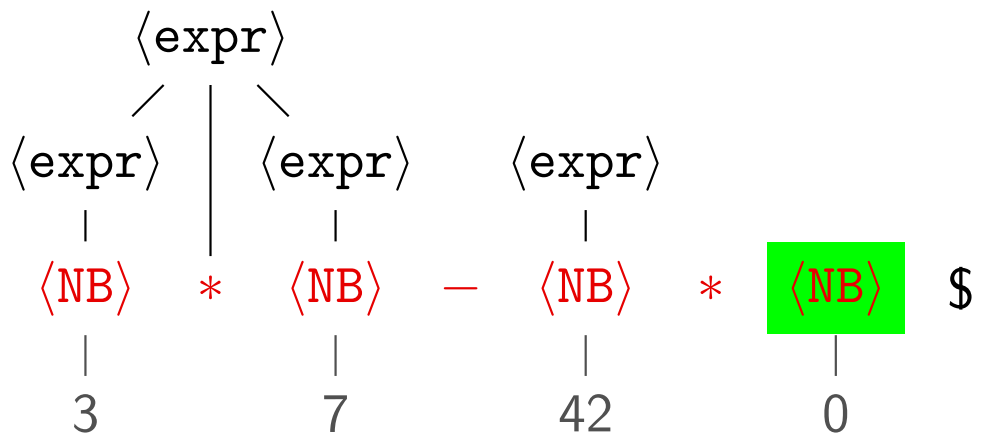
Automaton of the LR parser in motion



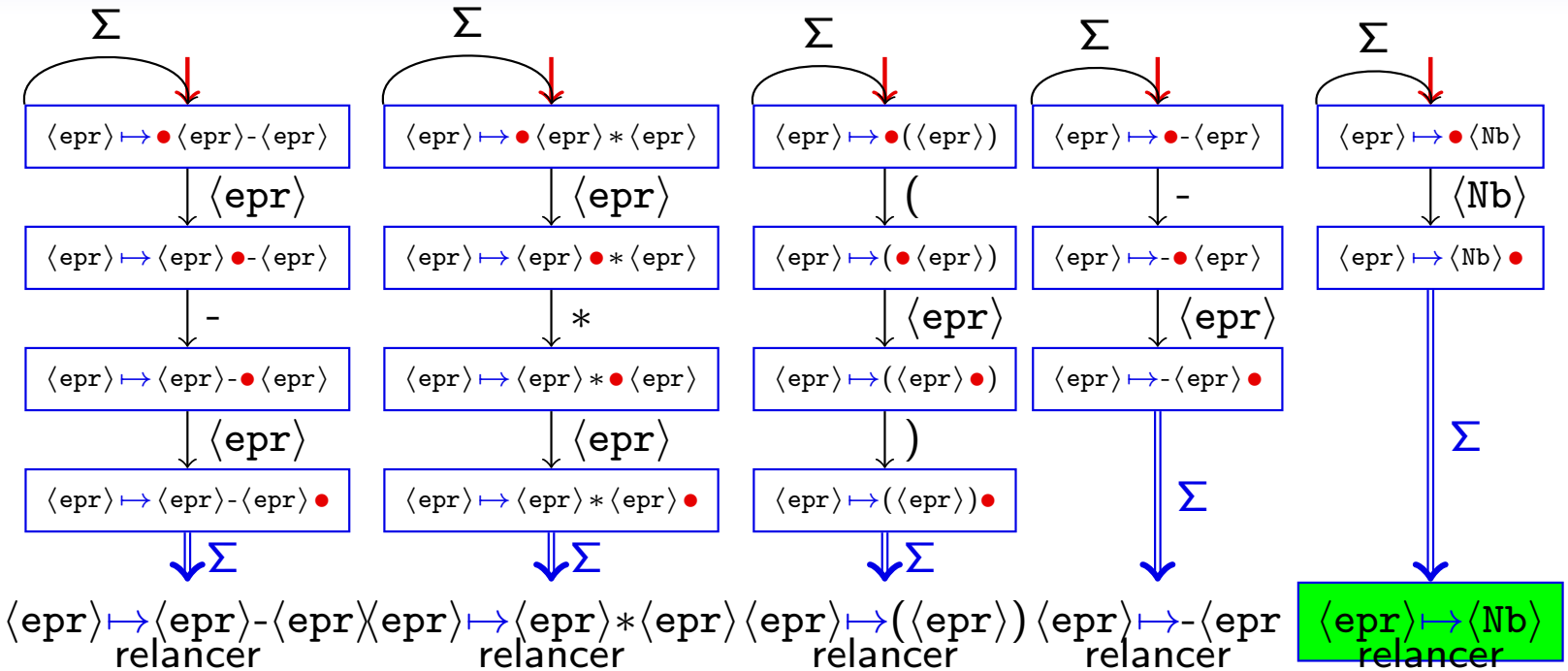


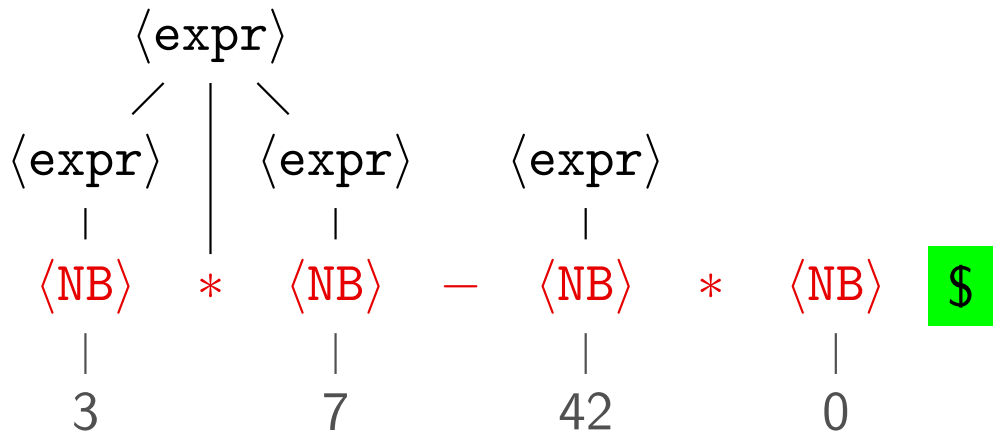
Automaton of the LR parser in motion



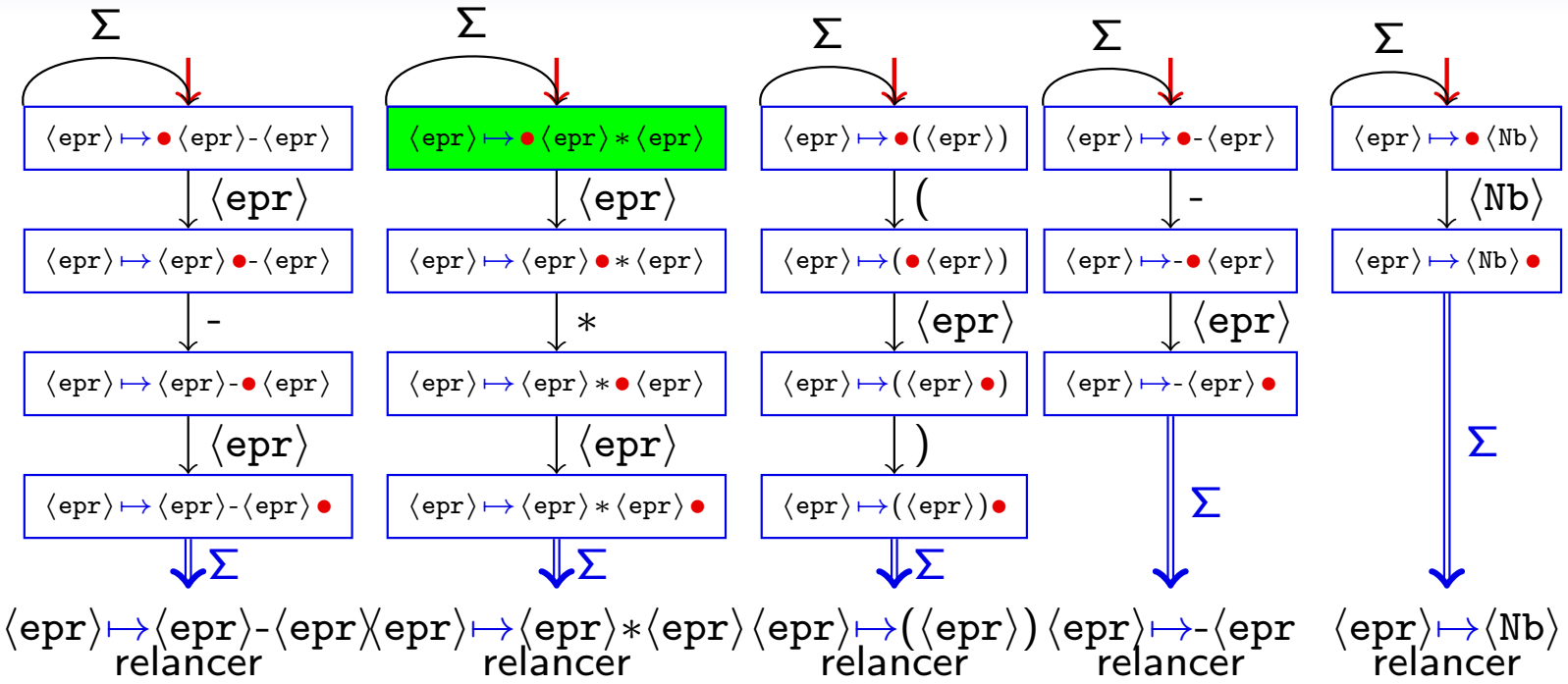


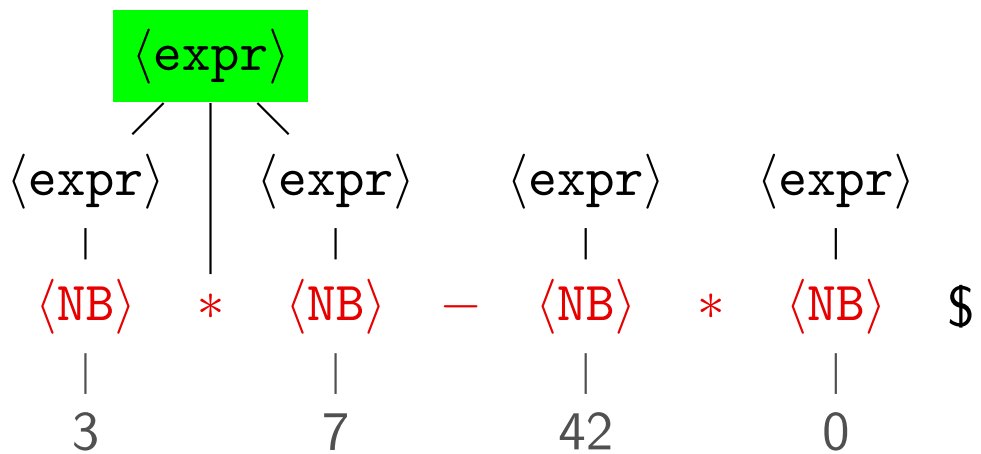
Automaton of the LR parser in motion



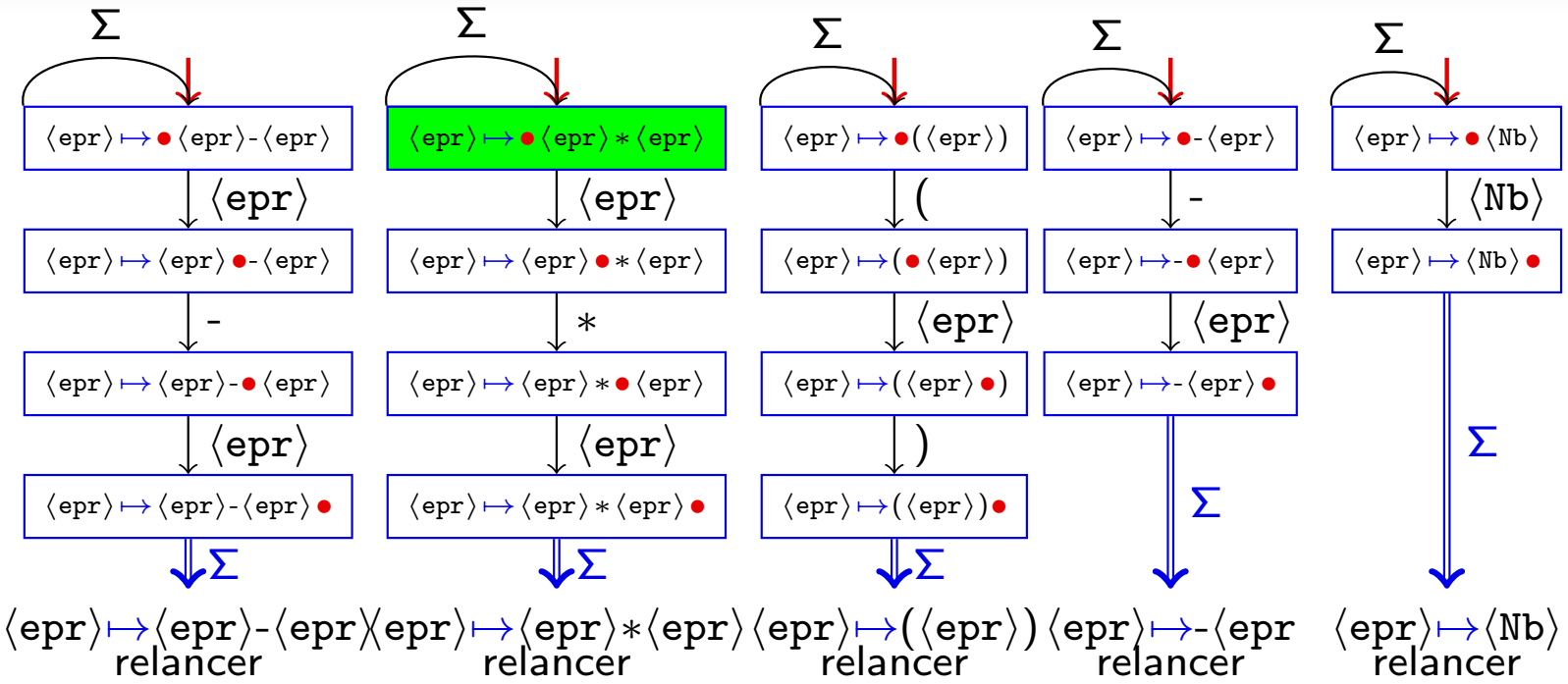


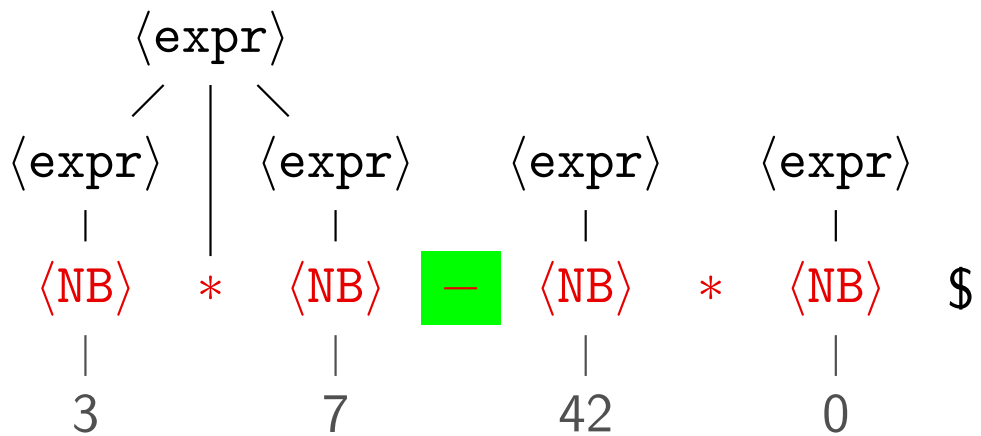
Automaton of the LR parser in motion



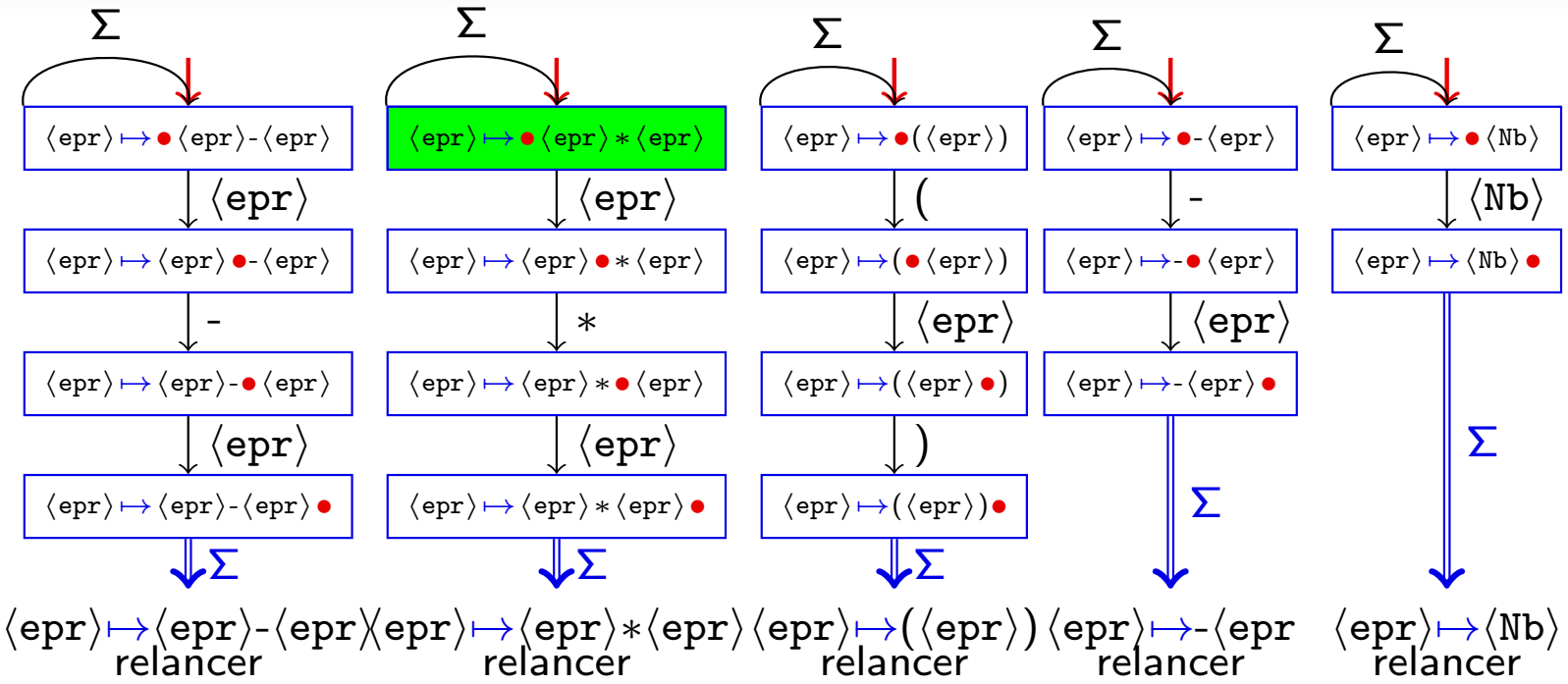


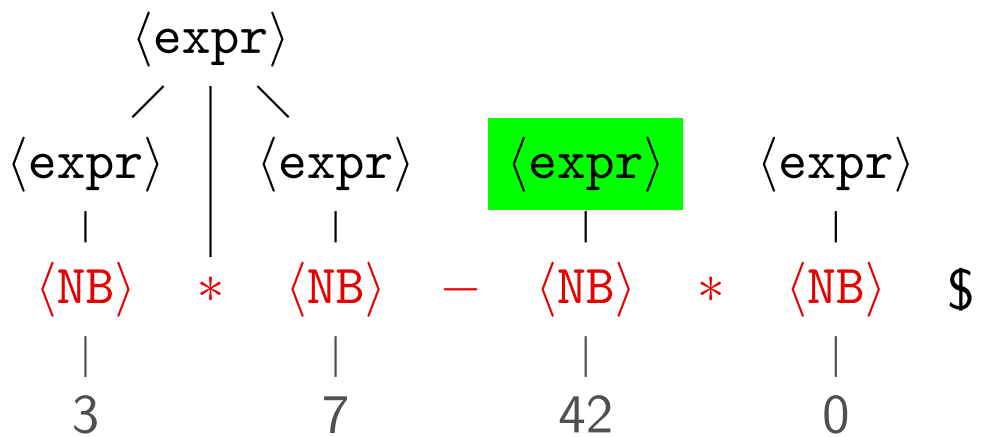
Automaton of the LR parser in motion



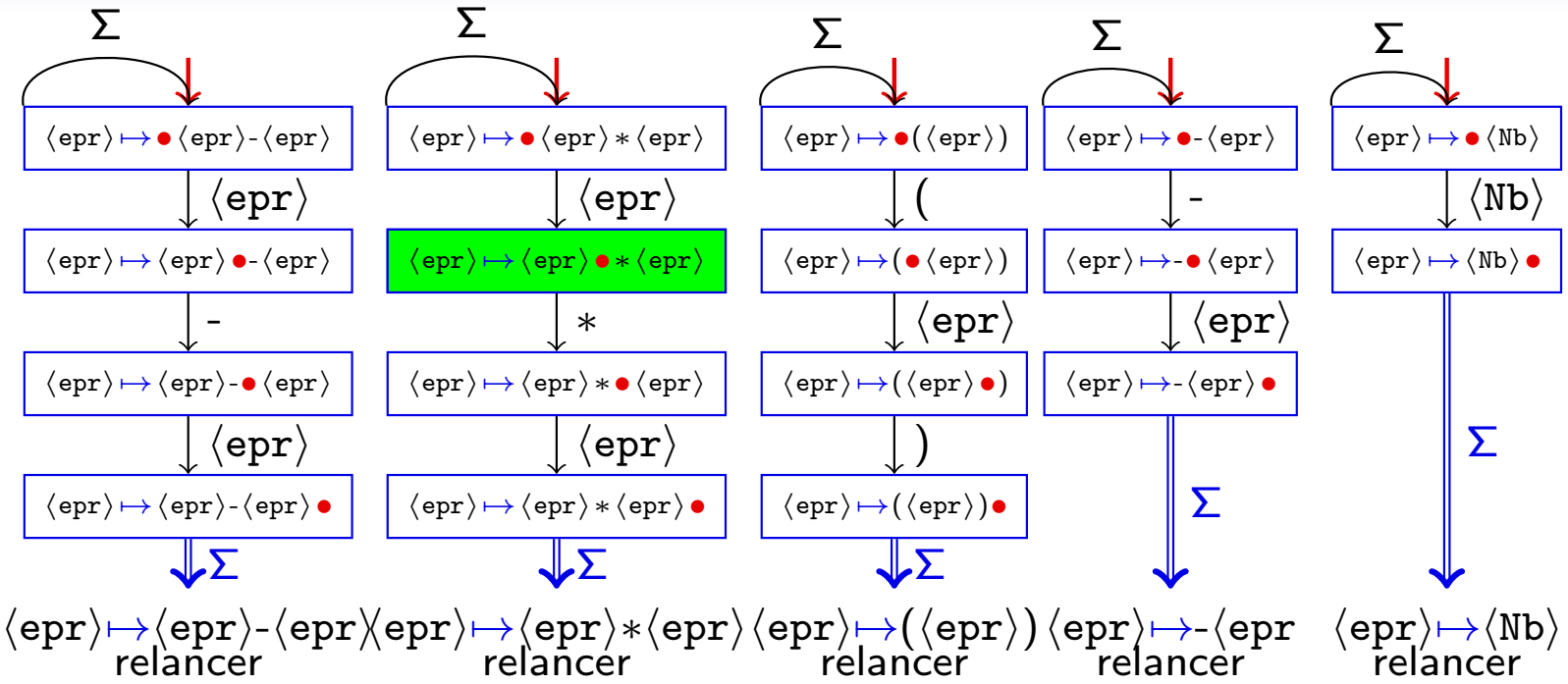


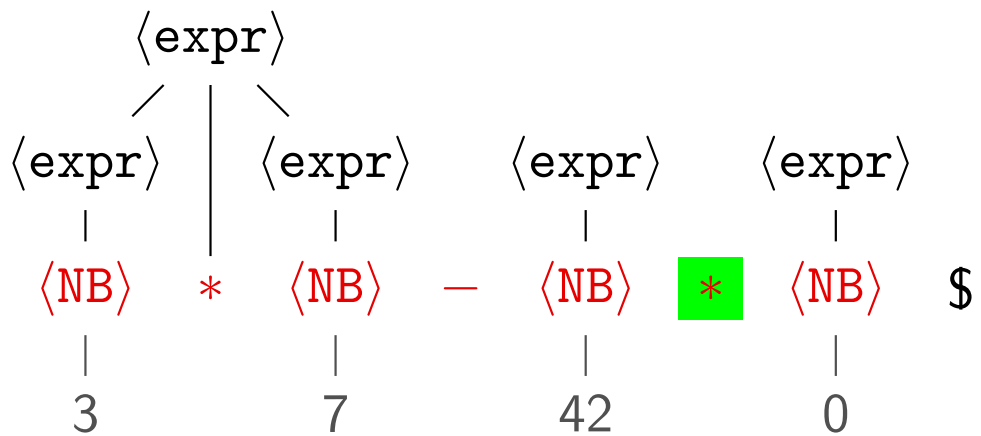
Automaton of the LR parser in motion



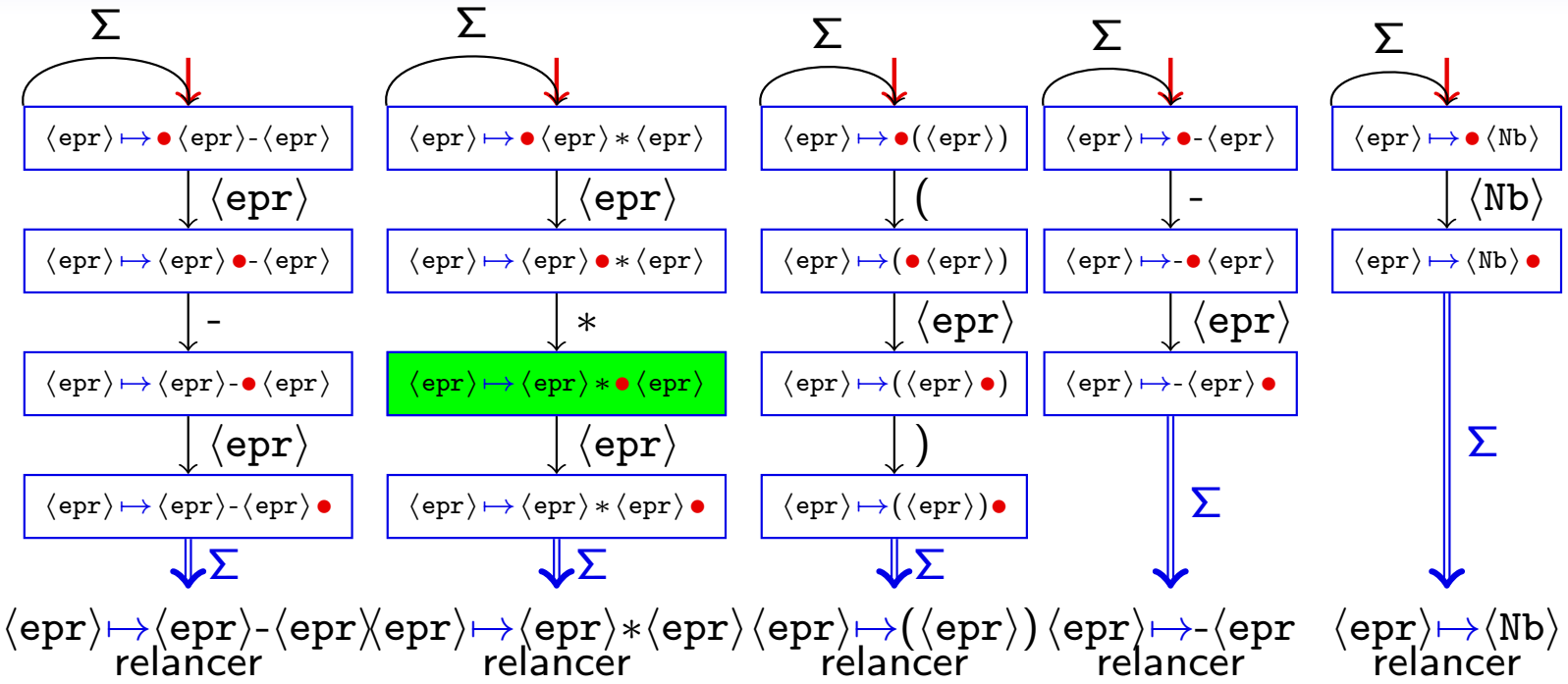


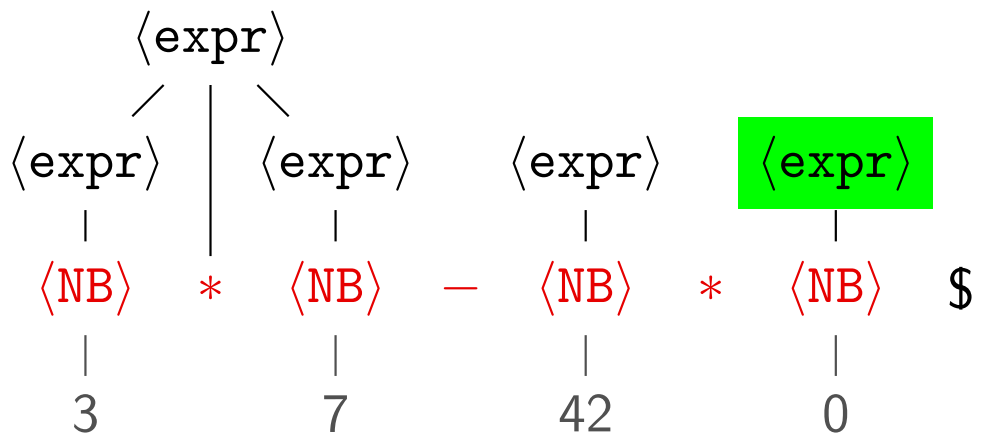
Automaton of the LR parser in motion



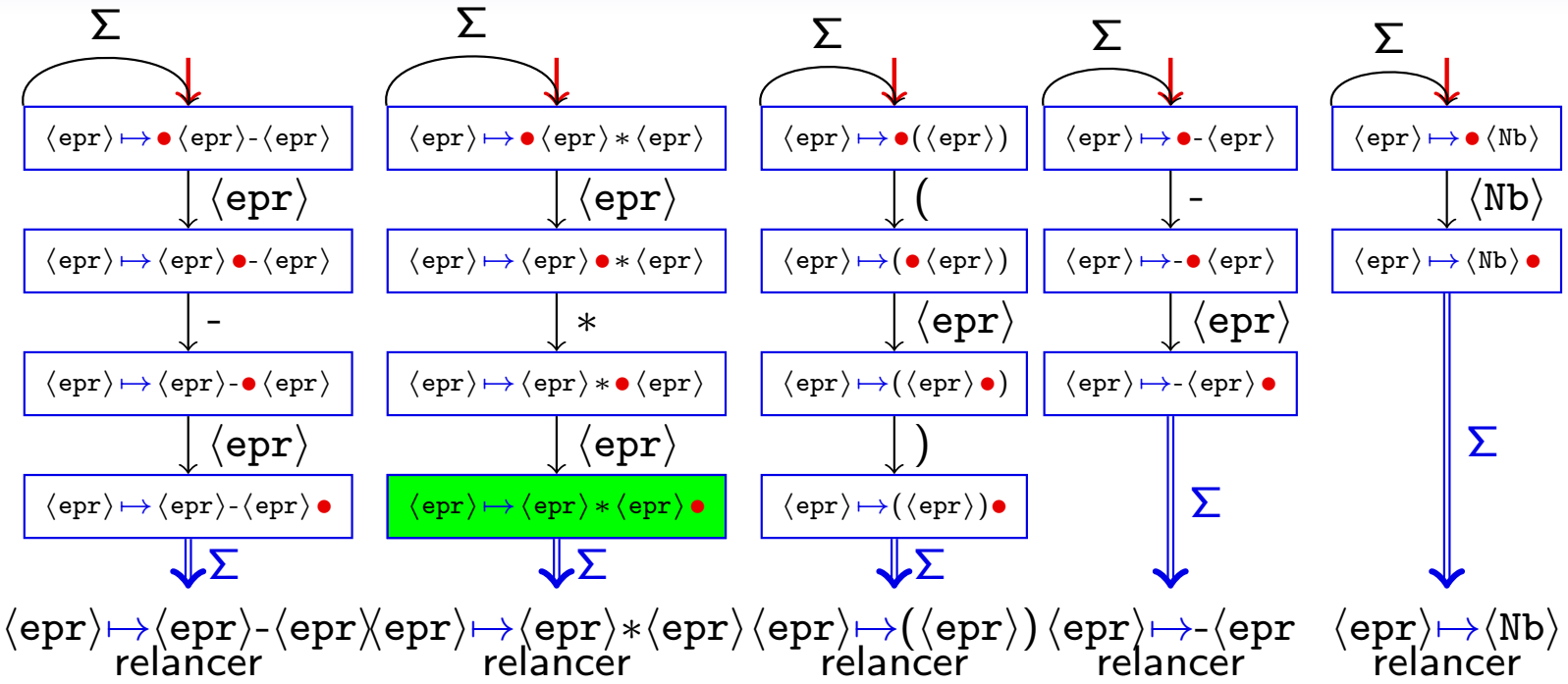


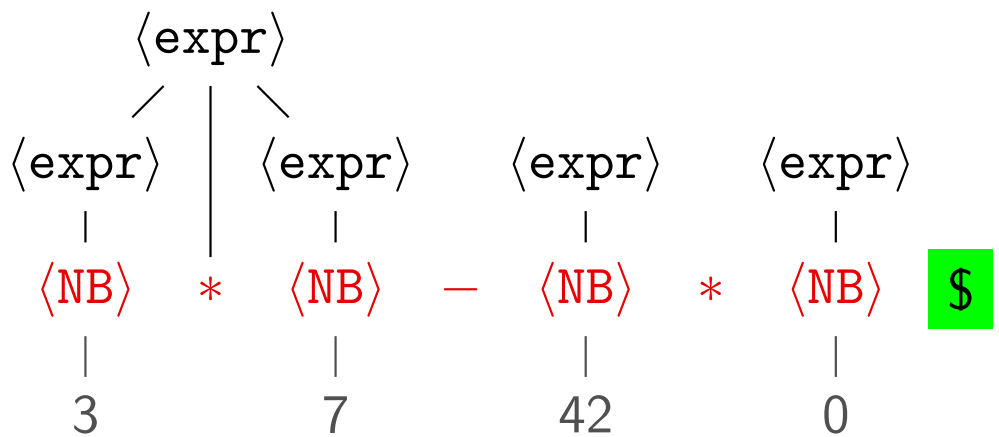
Automaton of the LR parser in motion



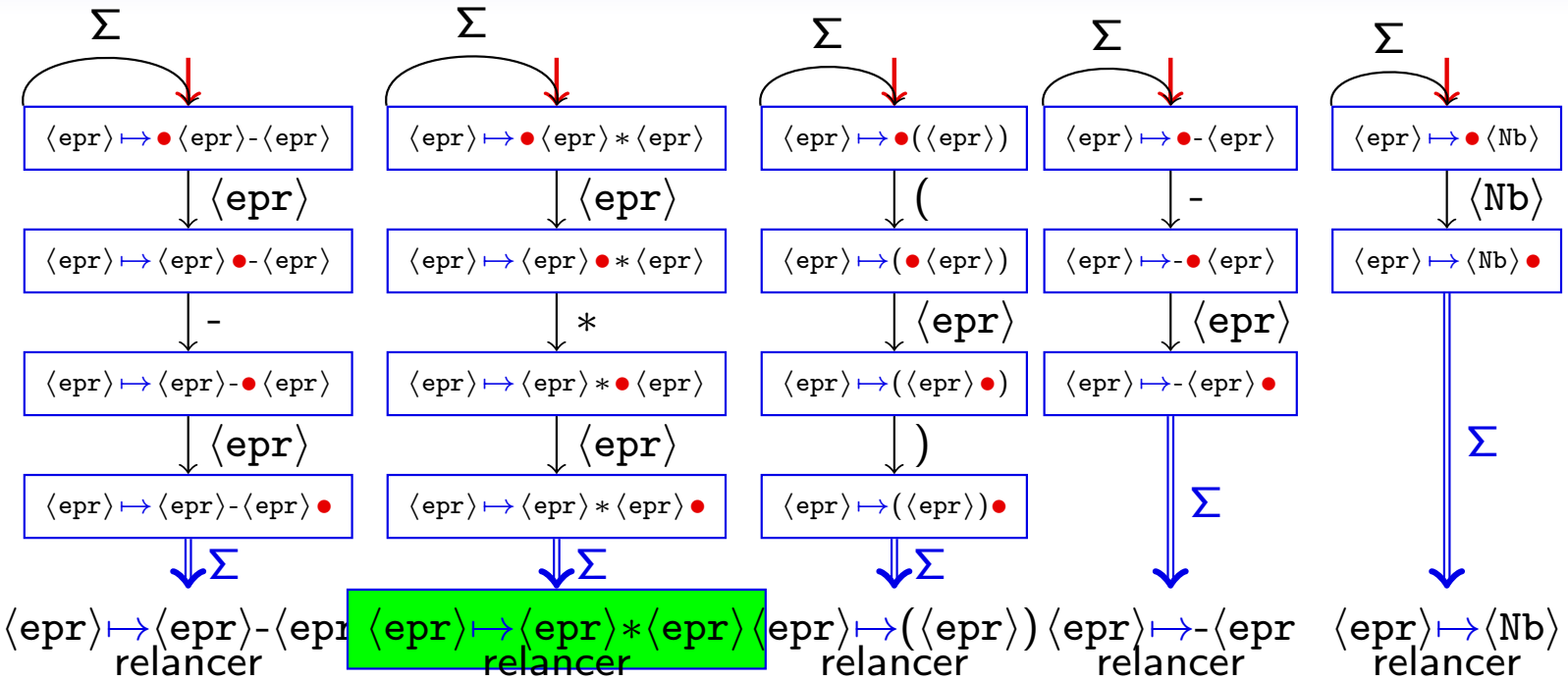


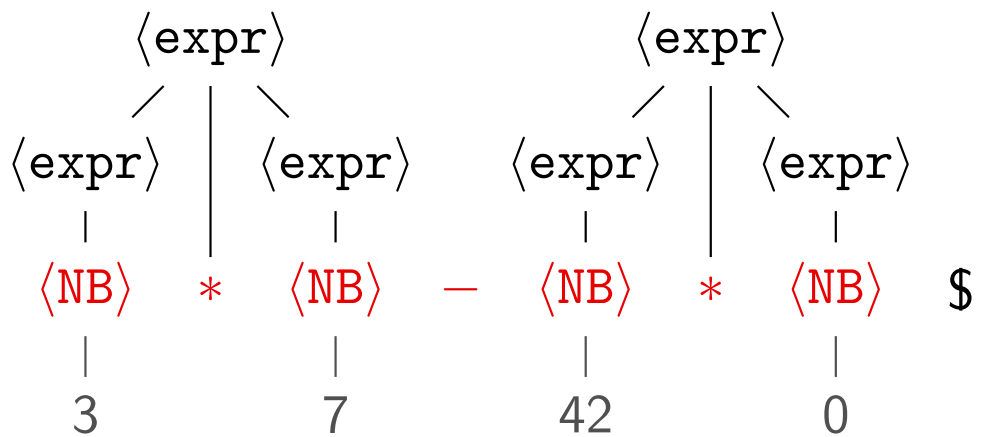
Automaton of the LR parser in motion



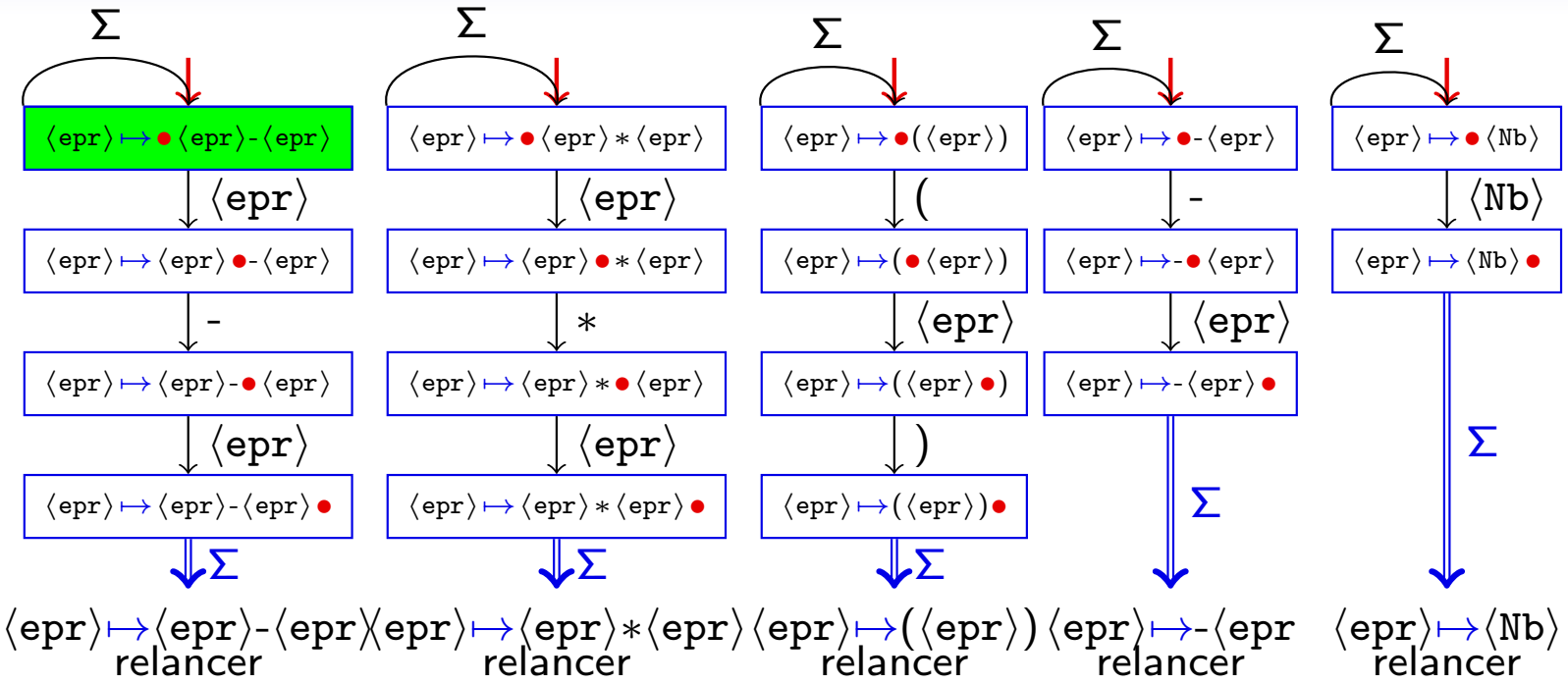


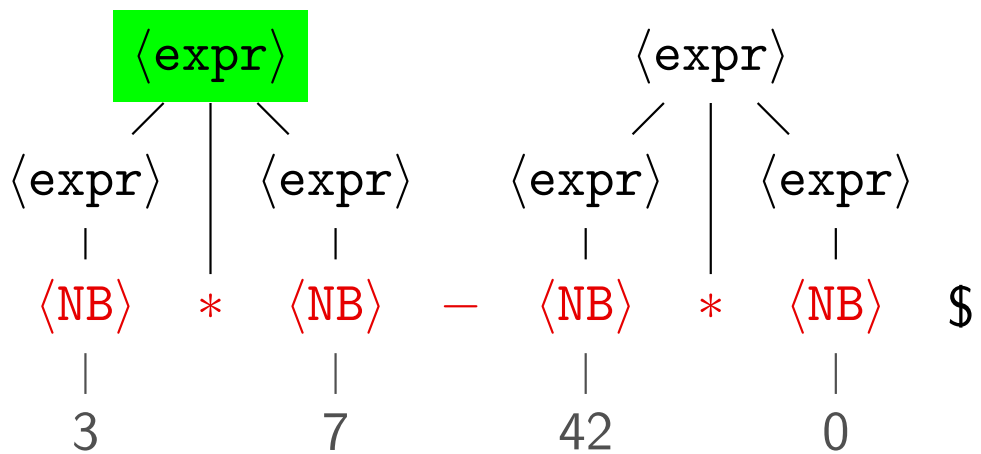
Automaton of the LR parser in motion



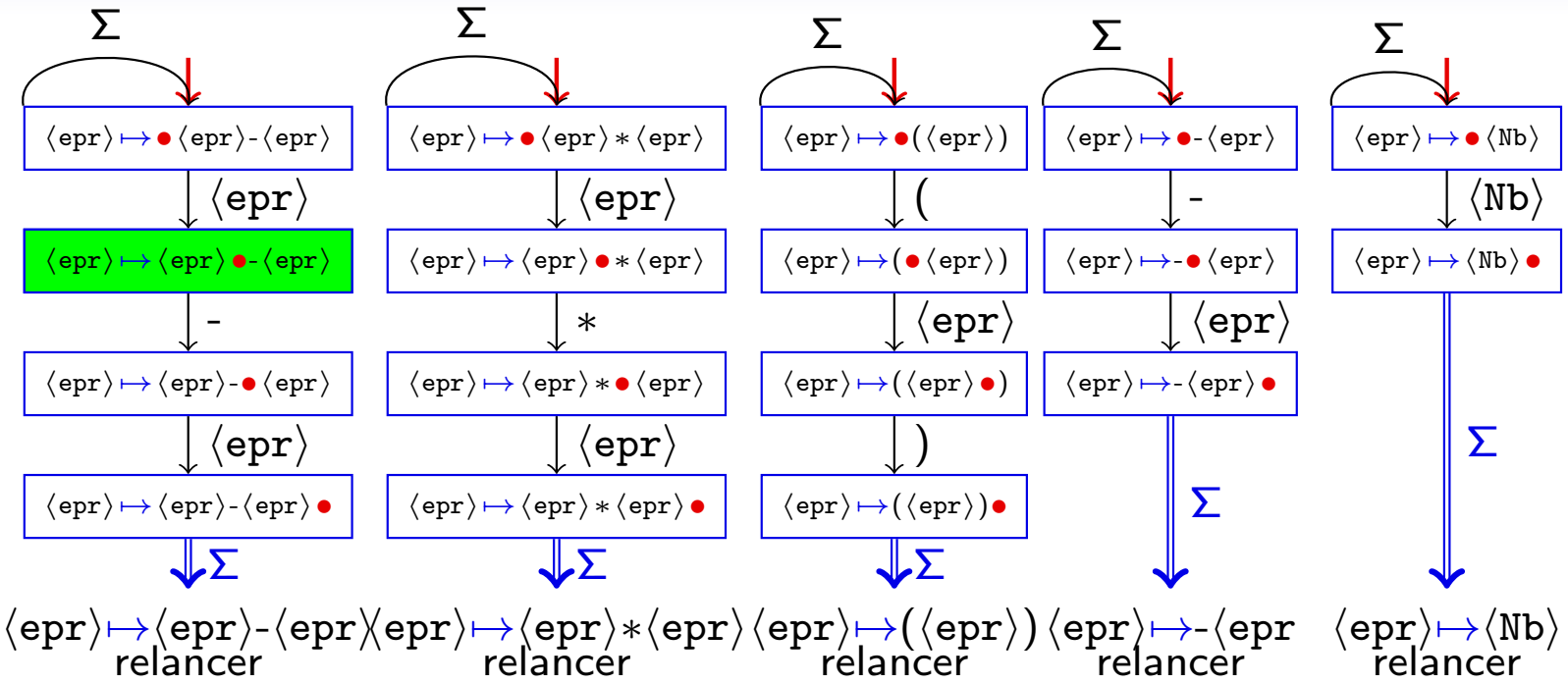


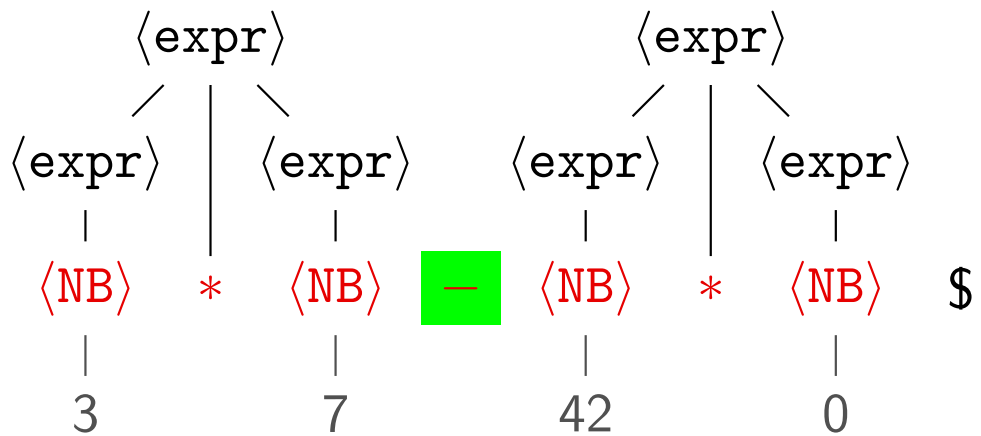
Automaton of the LR parser in motion



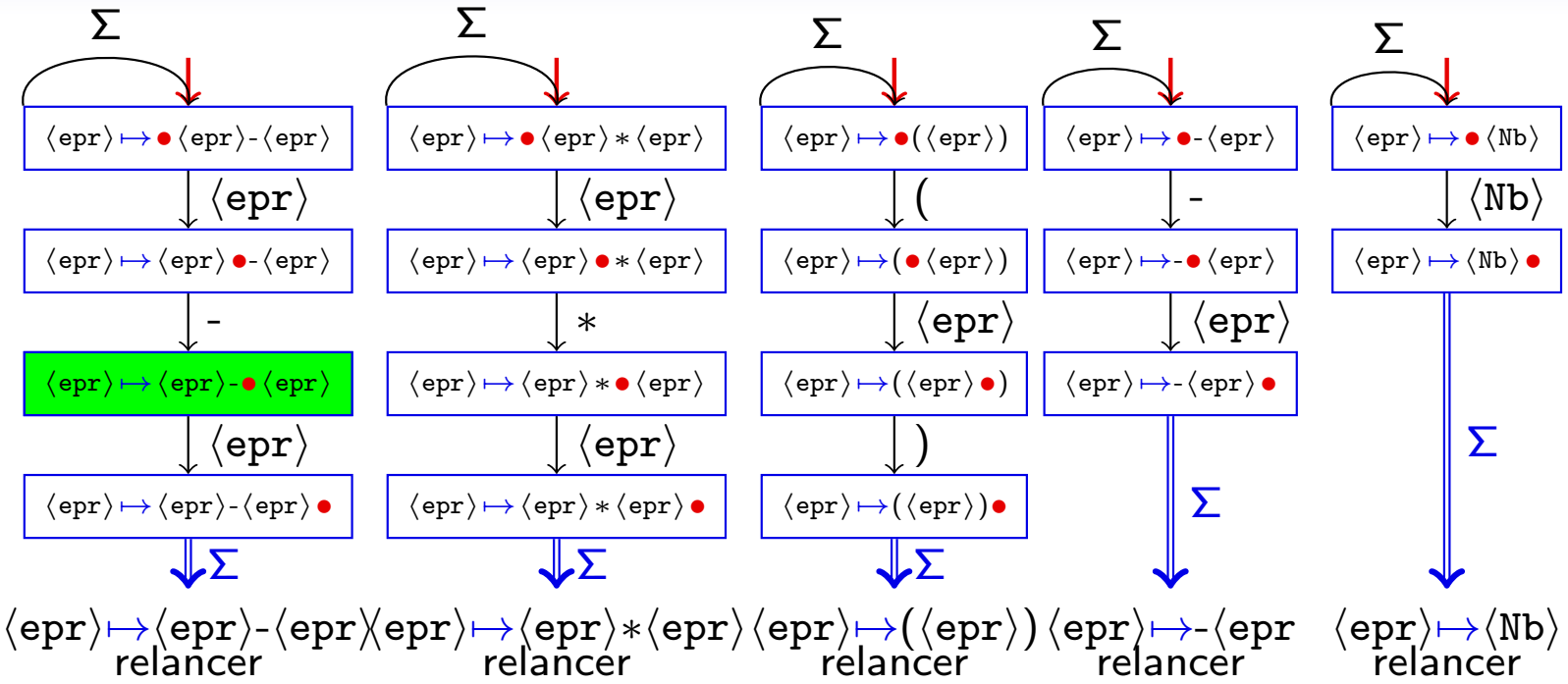


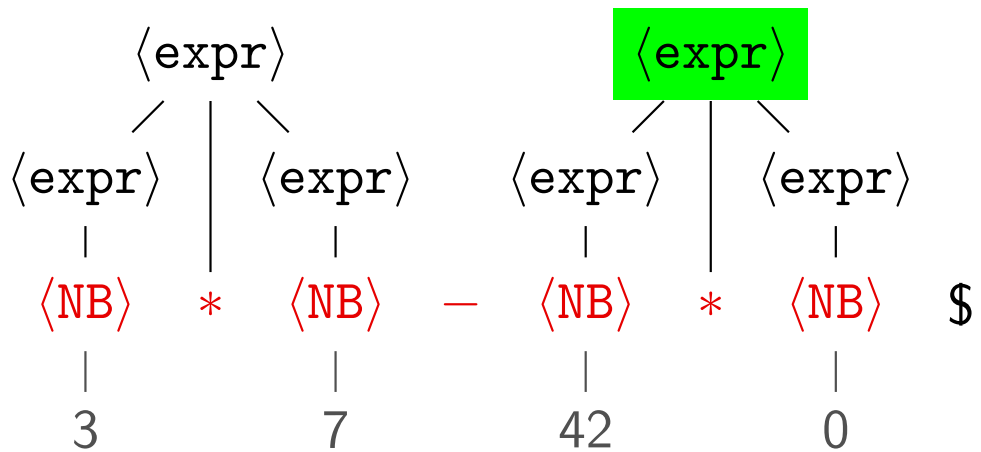
Automaton of the LR parser in motion



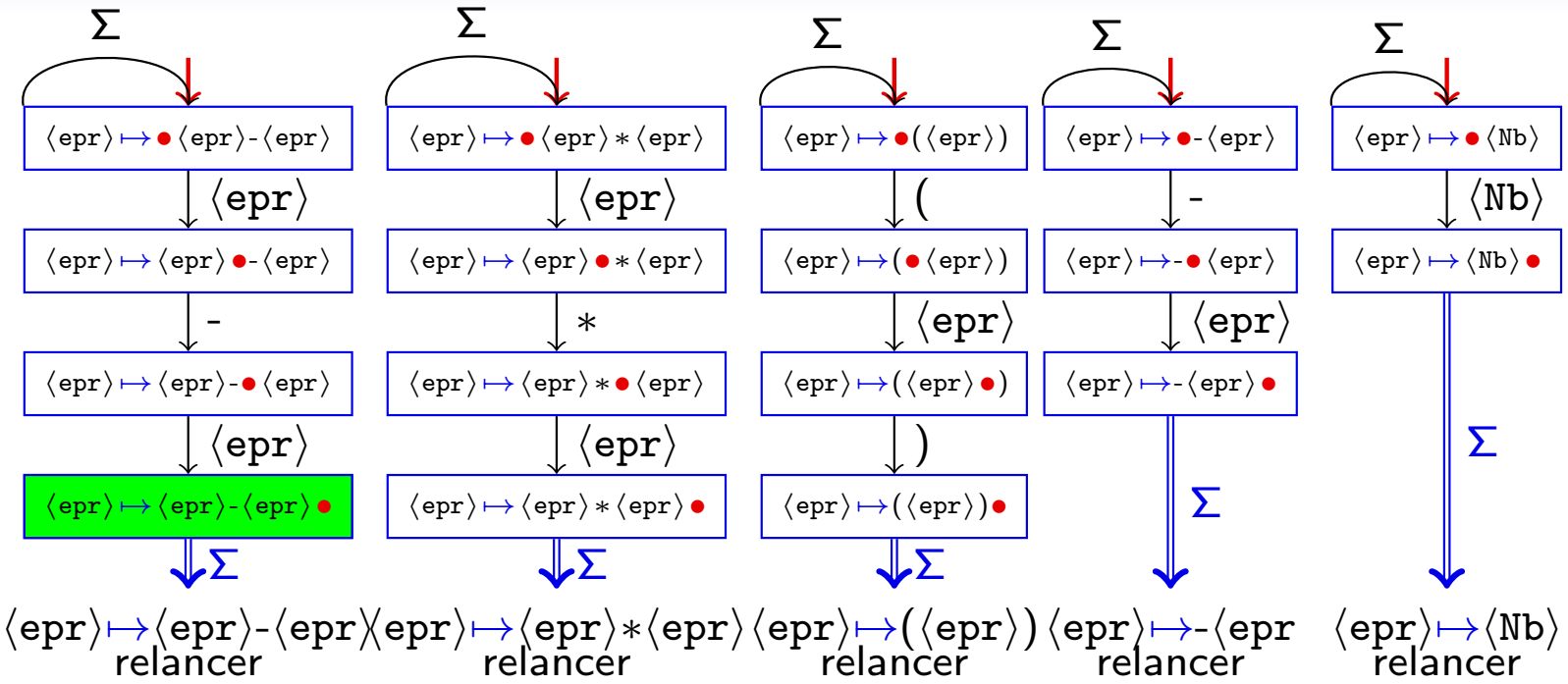


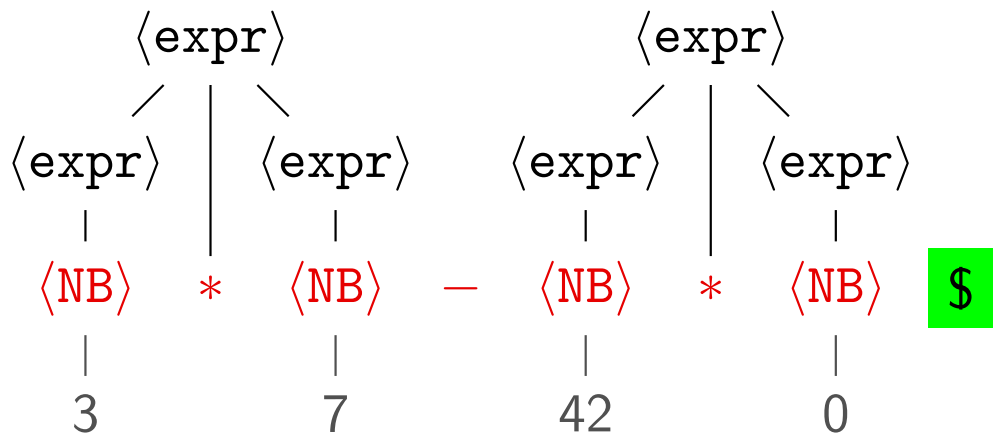
Automaton of the LR parser in motion



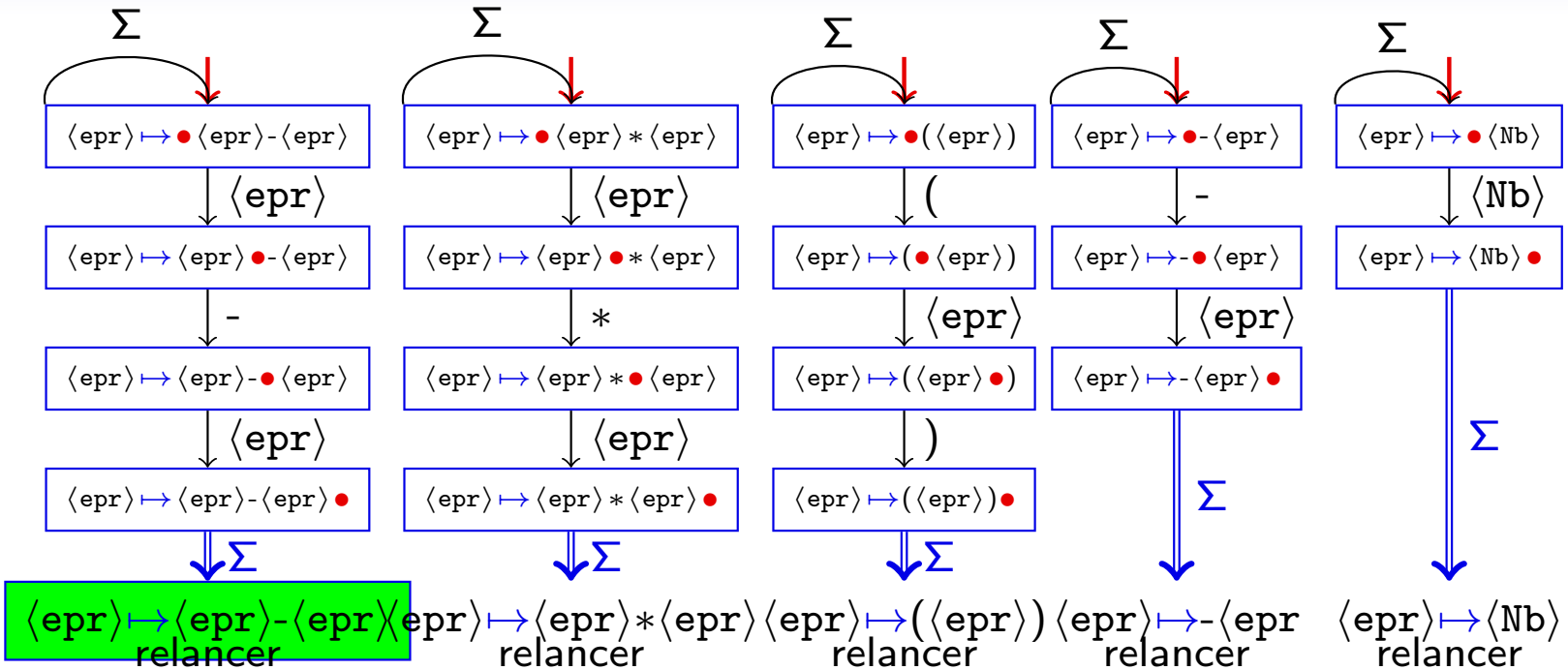


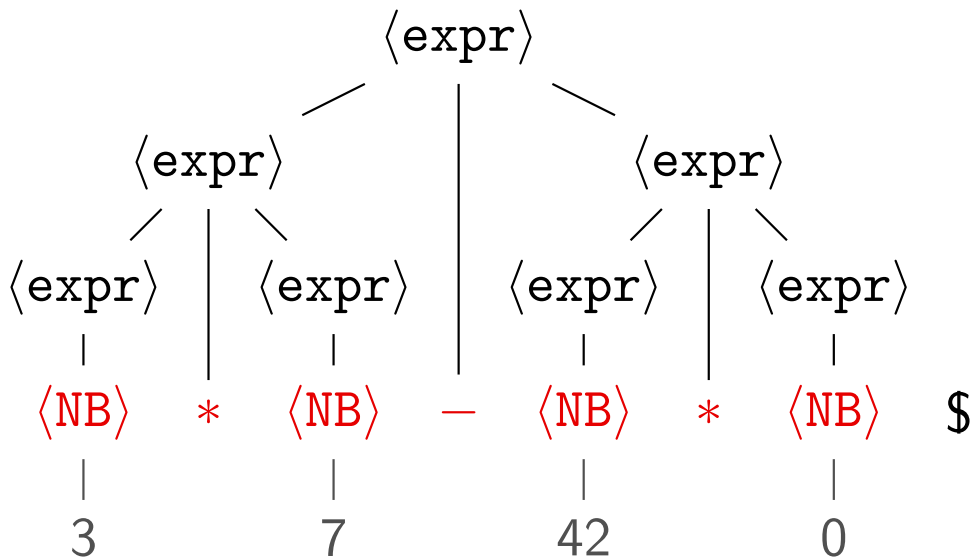
Automaton of the LR parser in motion



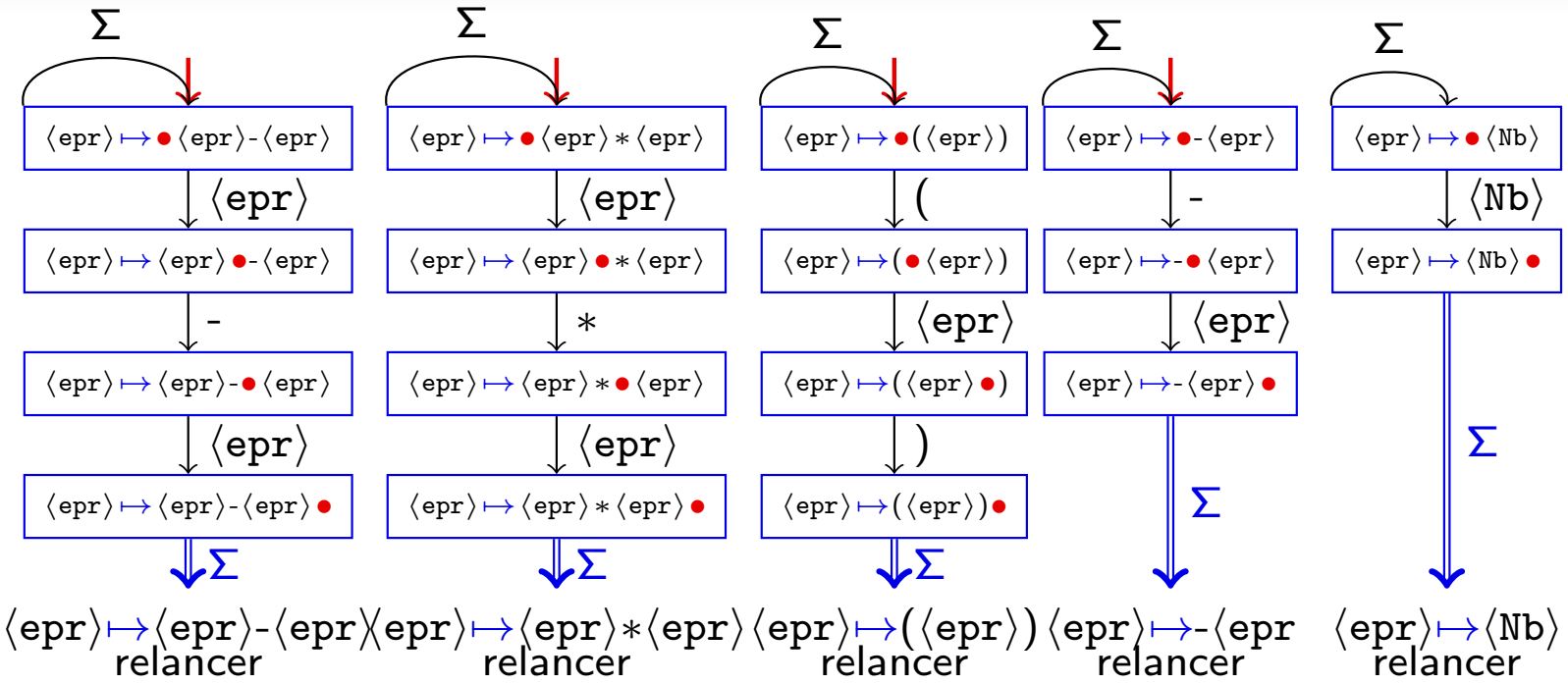


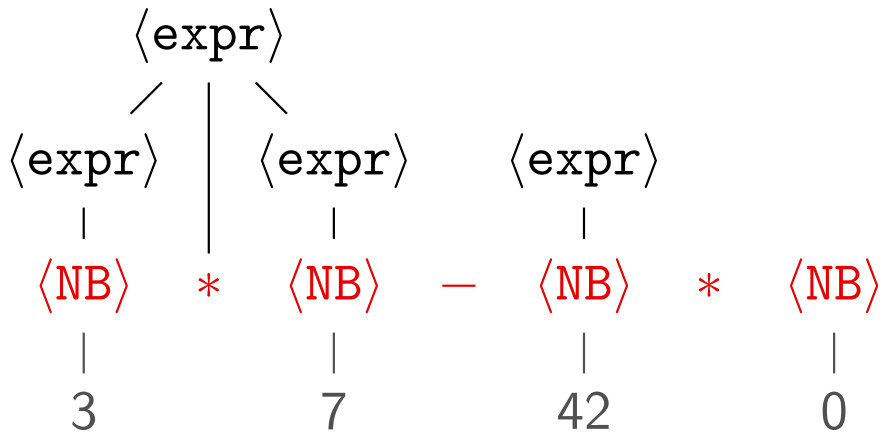
Automaton of the LR parser in motion



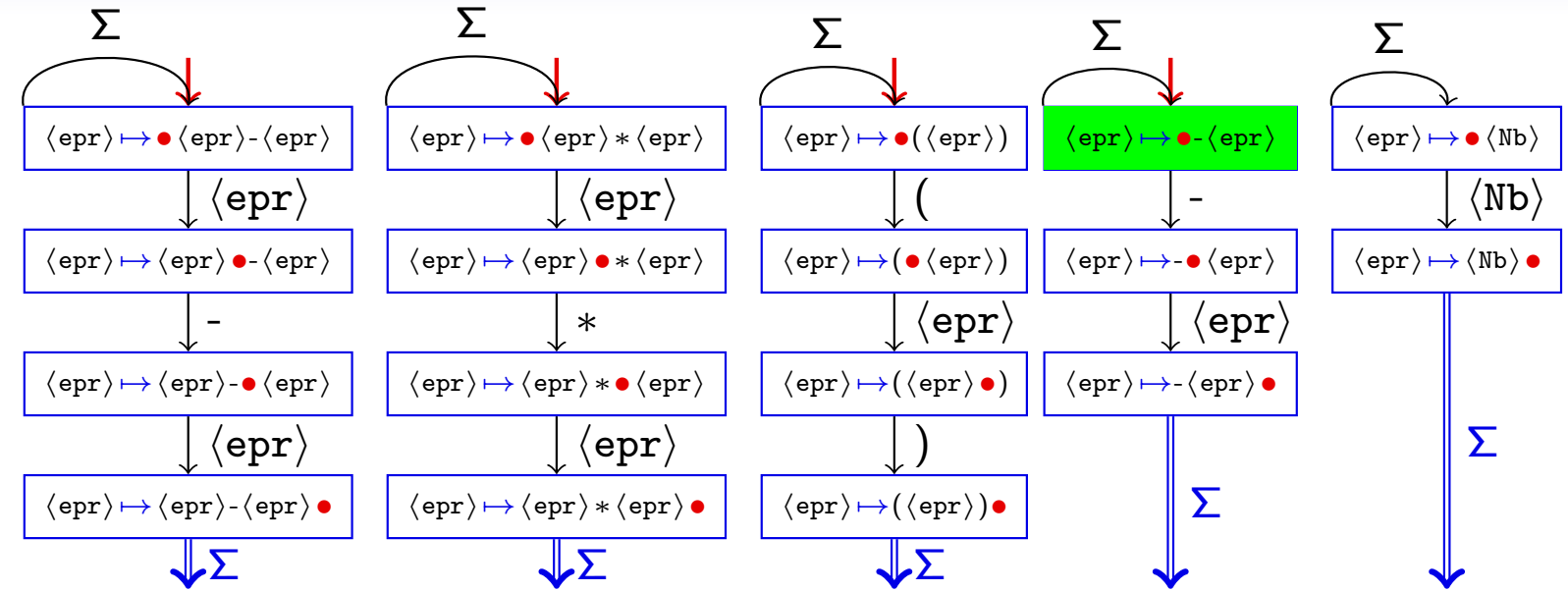


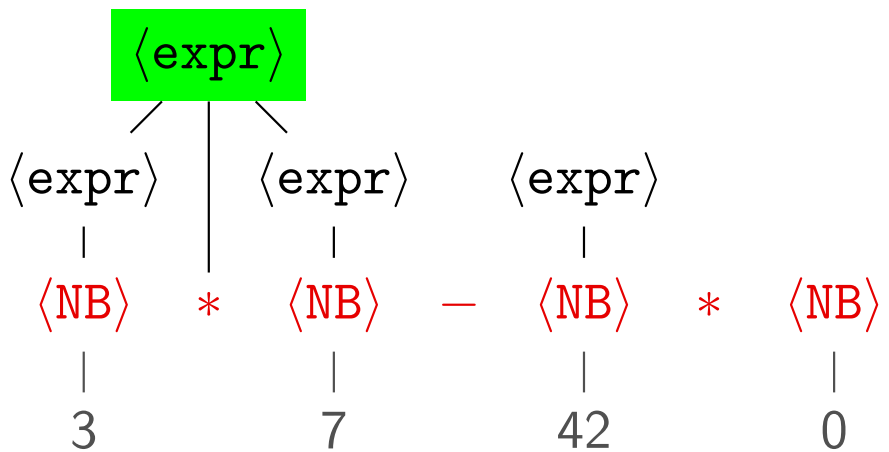
Failure of our attempt



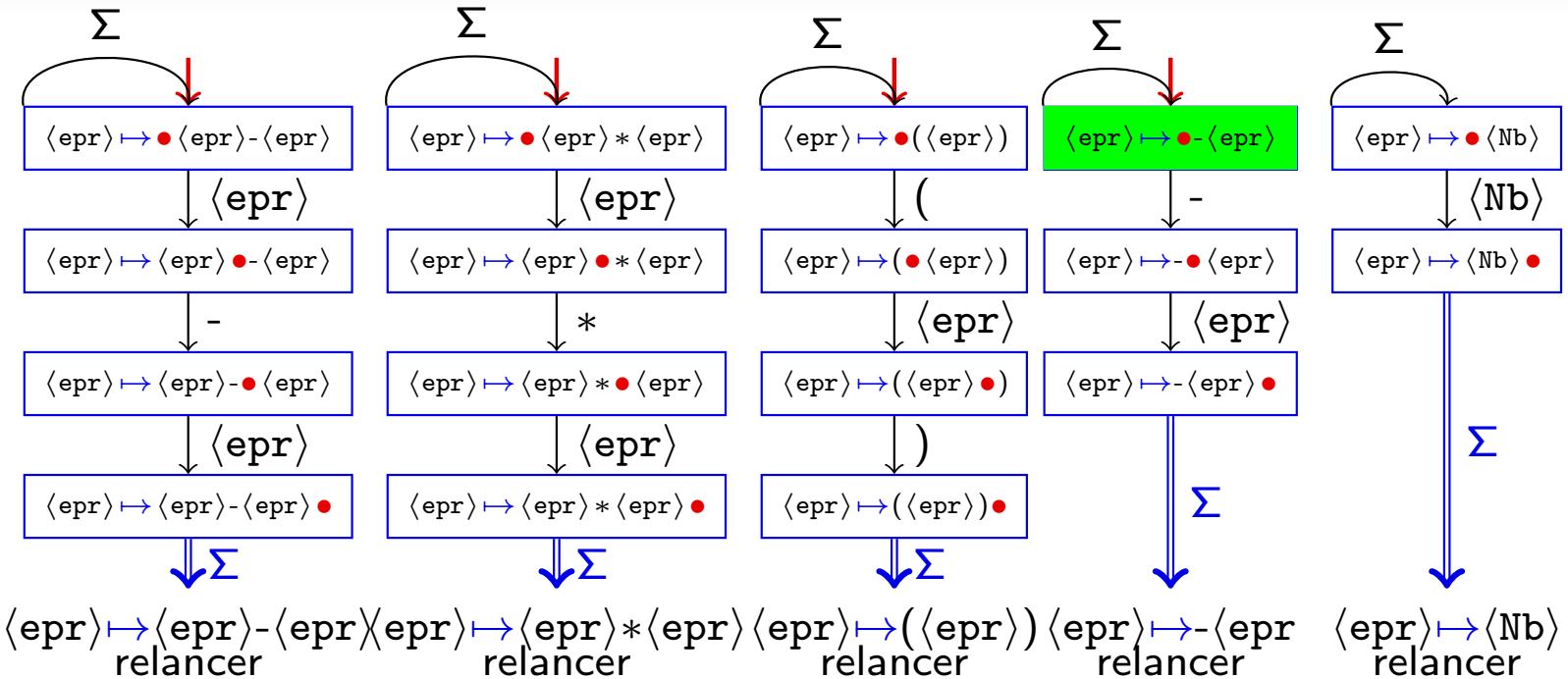


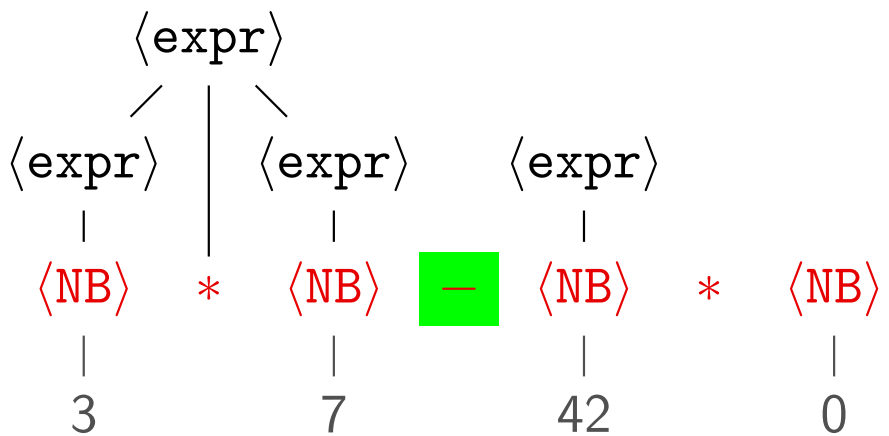
Failure of our attempt



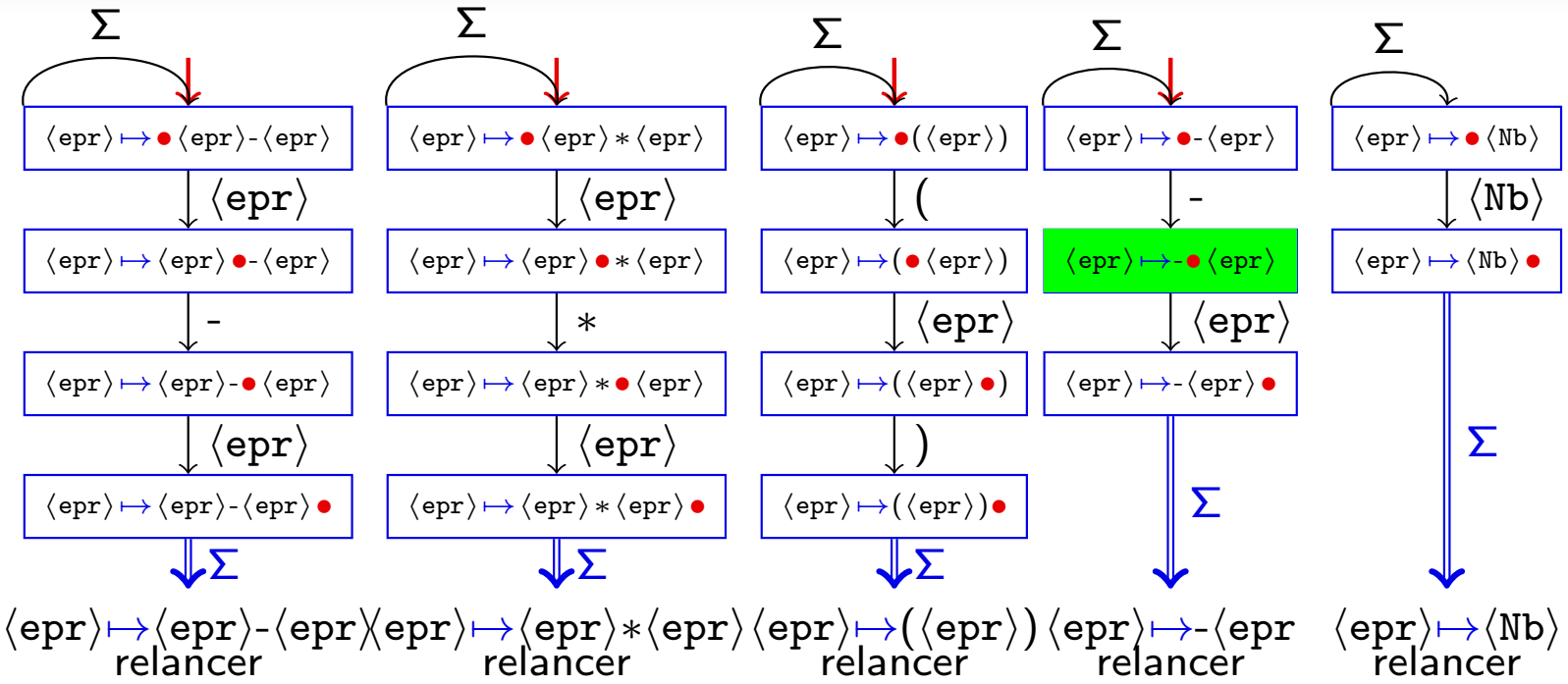


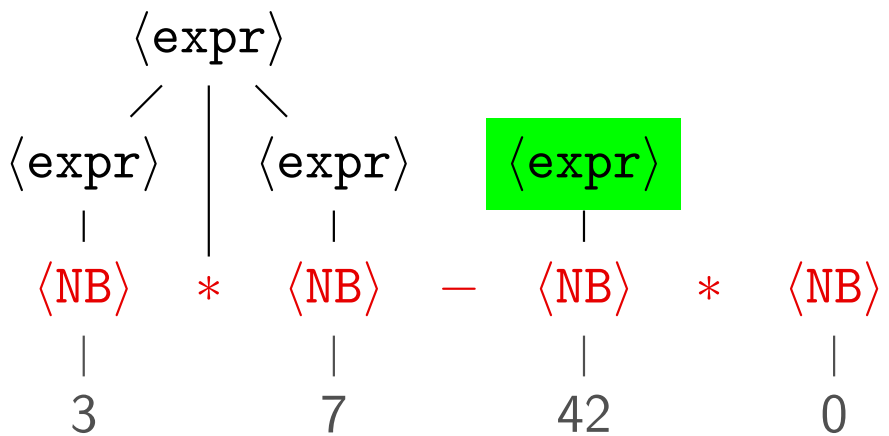
Failure of our attempt



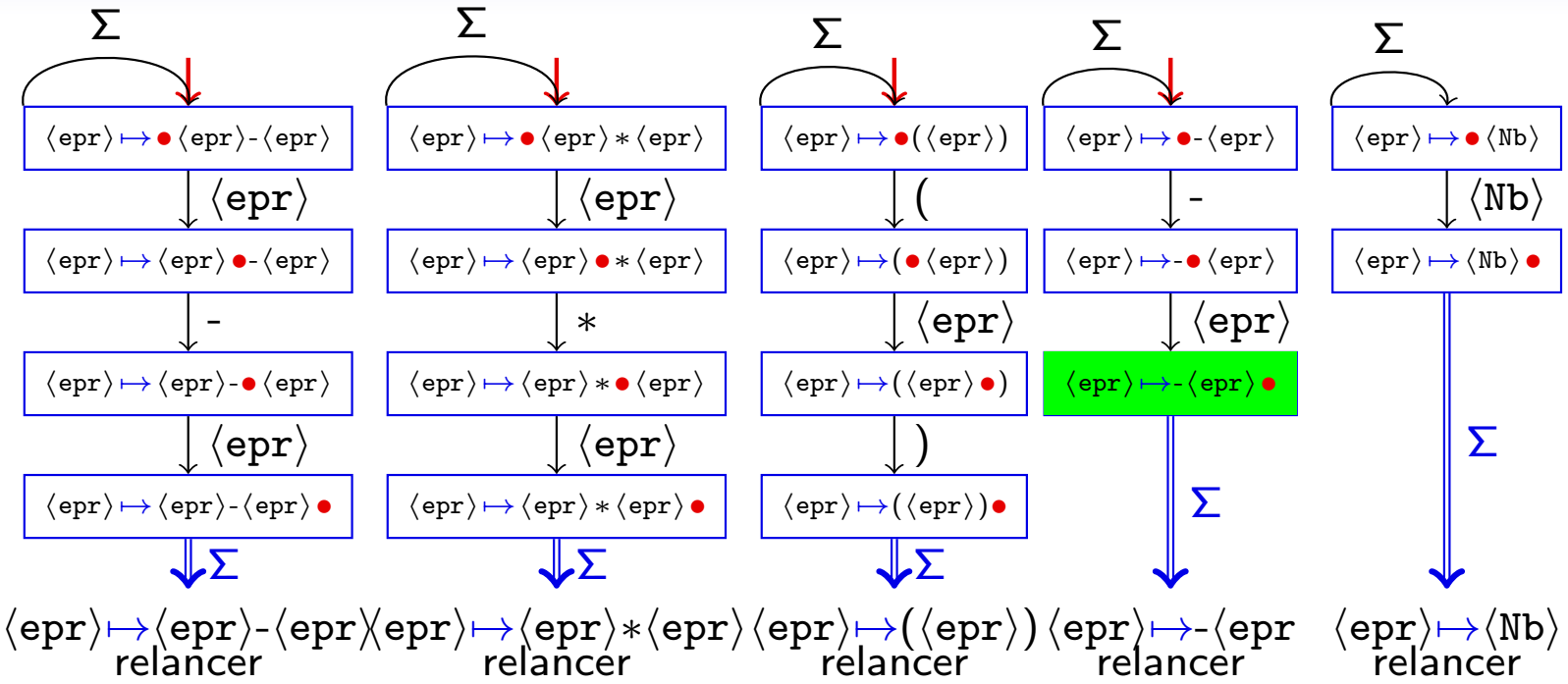


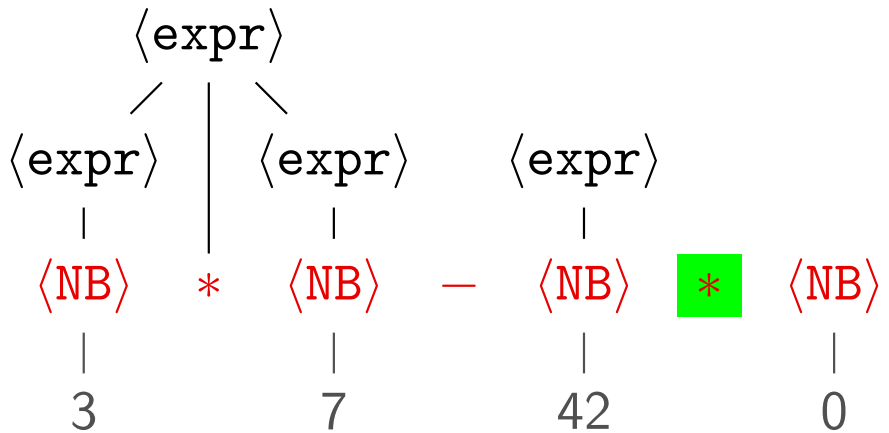
Failure of our attempt



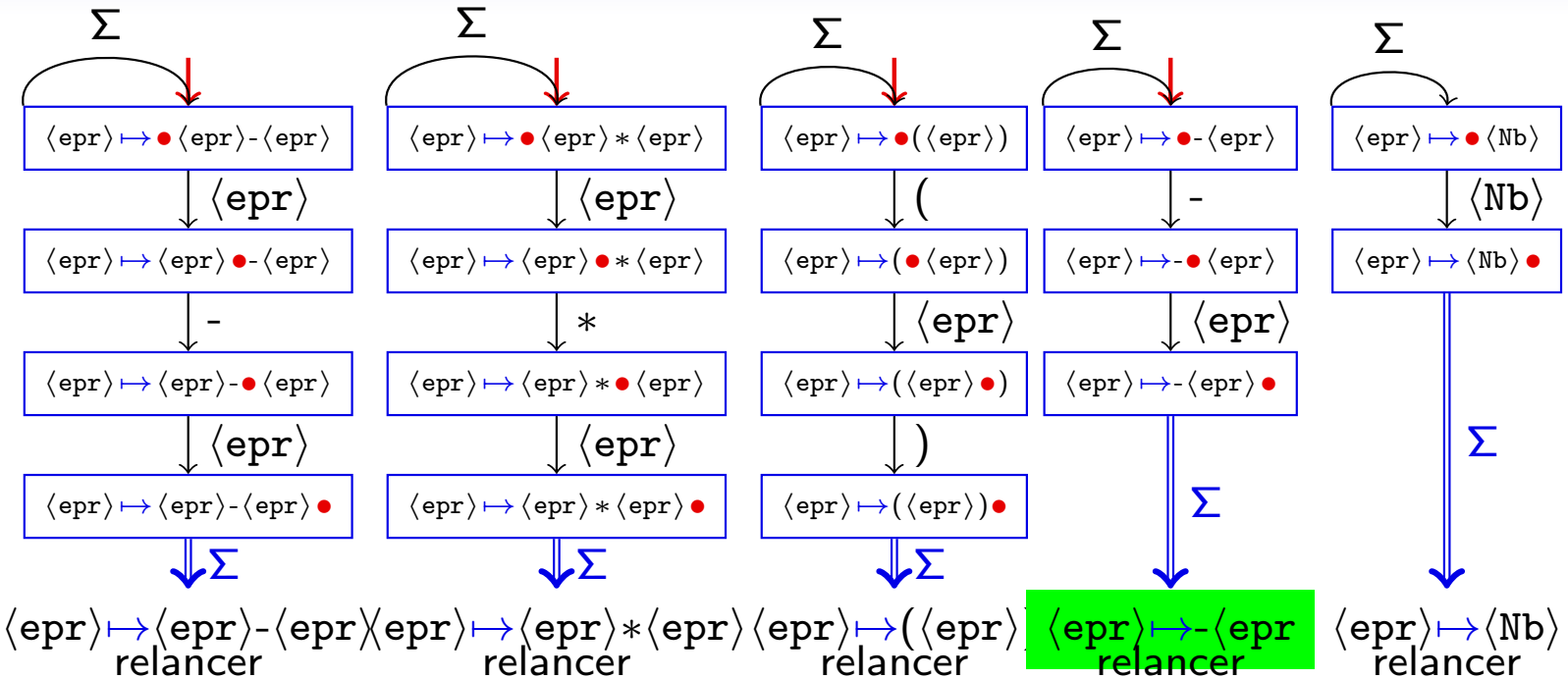


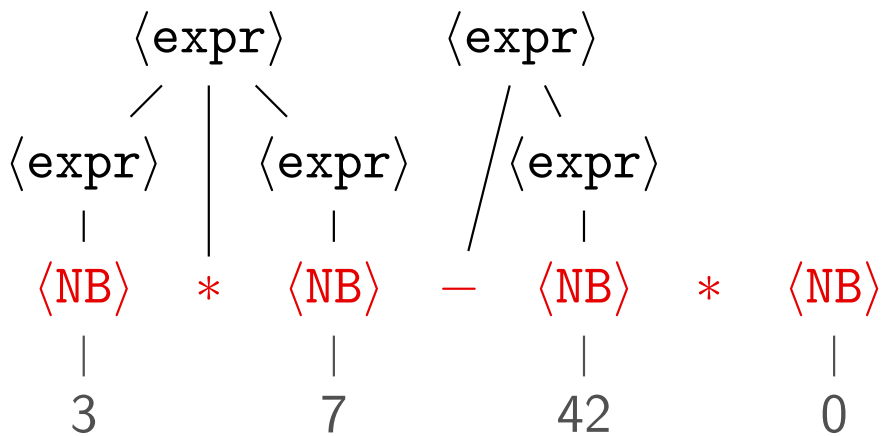
Failure of our attempt



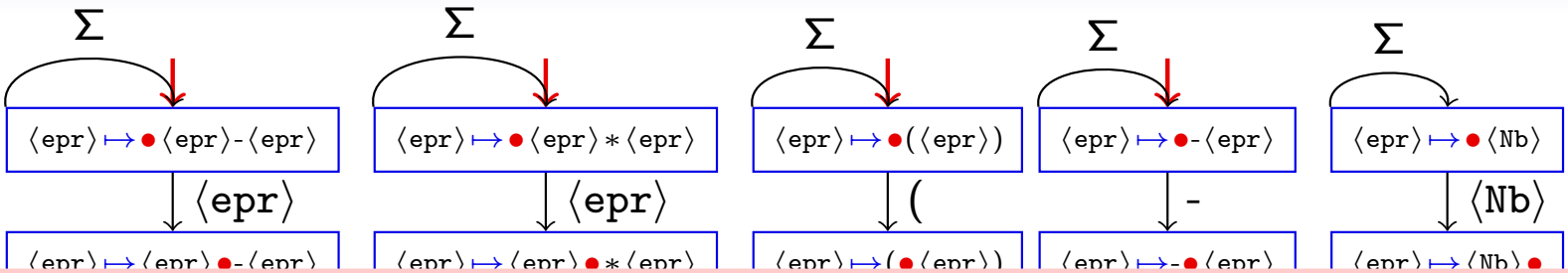


Failure of our attempt





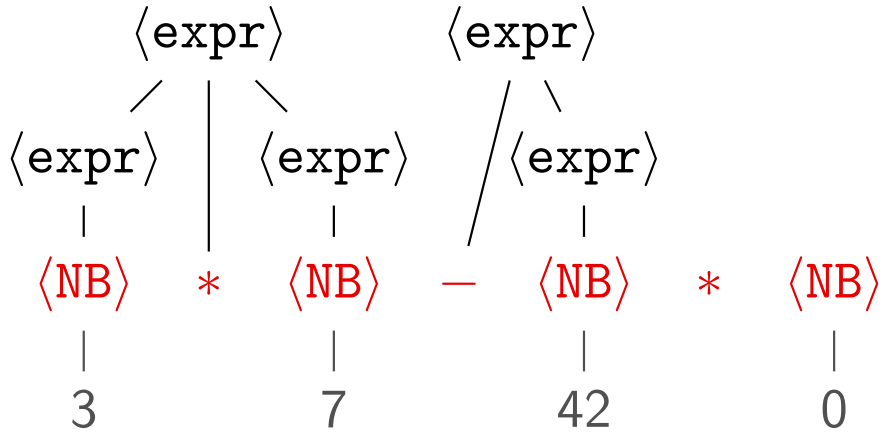
Failure of our attempt



Why is it critical ?
(Conflict resolutions may resolve this ?)

The unary '-' has precedence over everything else, it will thus be selected before everything else in any reasonable determinisation...

We can have similar issues with non-ambiguous grammars.



What can we do ???
Force the more elements in the ND automaton

Non-terminal are to be created when last seen.

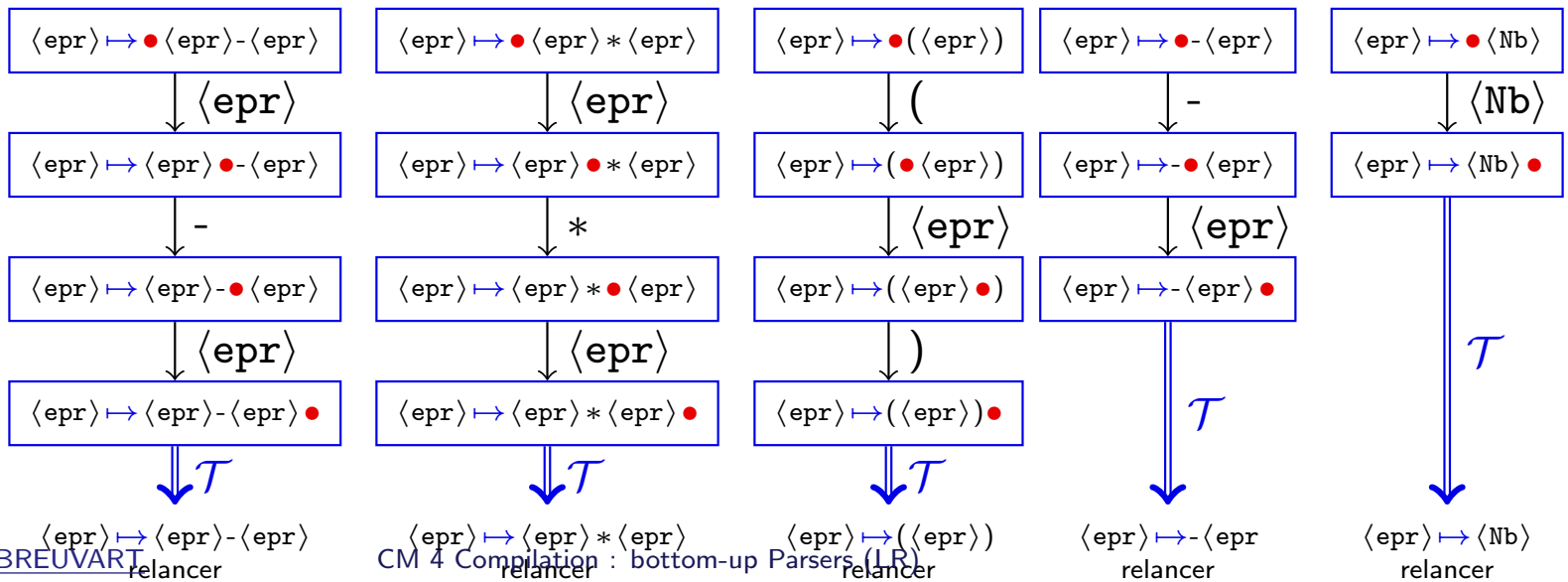
Thus there is no way any non-terminal exist after the point of an action.
↳ Actions are only taken on terminal peak.

Only creat non terminal where we need them

Do not "test" each rule each step with a Σ-loop,
↳ try a rule where we may need the created non-terminal.

Second attempt

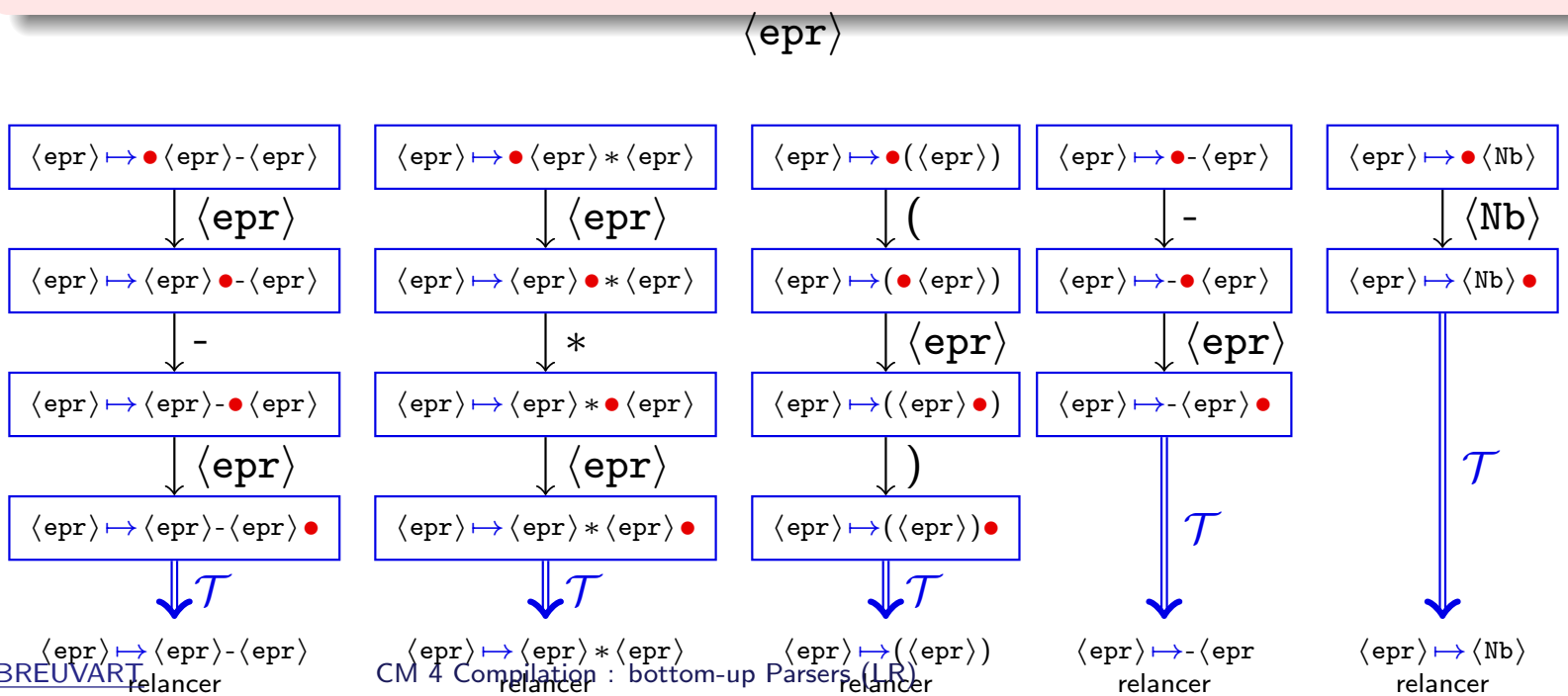
Actions taken on terminal peak are easy to deal with, but how to try a rule where we may need the created NT



Second attempt

Actions taken on terminal peak are easy to deal with, but how to try a rule where we may need the created NT

- add a dumy state for each non-terminal N,
  meaning : "I neet to create a N"

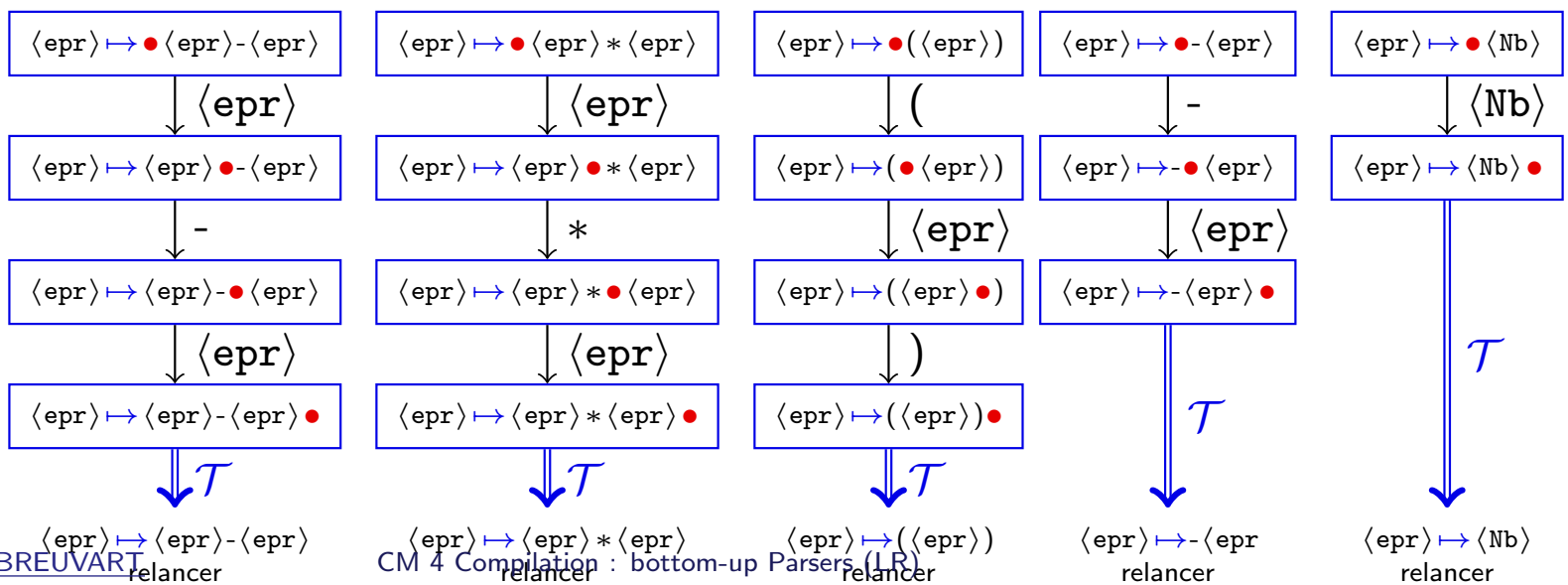


Second attempt

Actions taken on terminal peak are easy to deal with, but how to try a rule where we may need the created NT

- add a dumy state for each non-terminal N, meaning : "I neet to create a N"
- only the pricipal one S is initial,

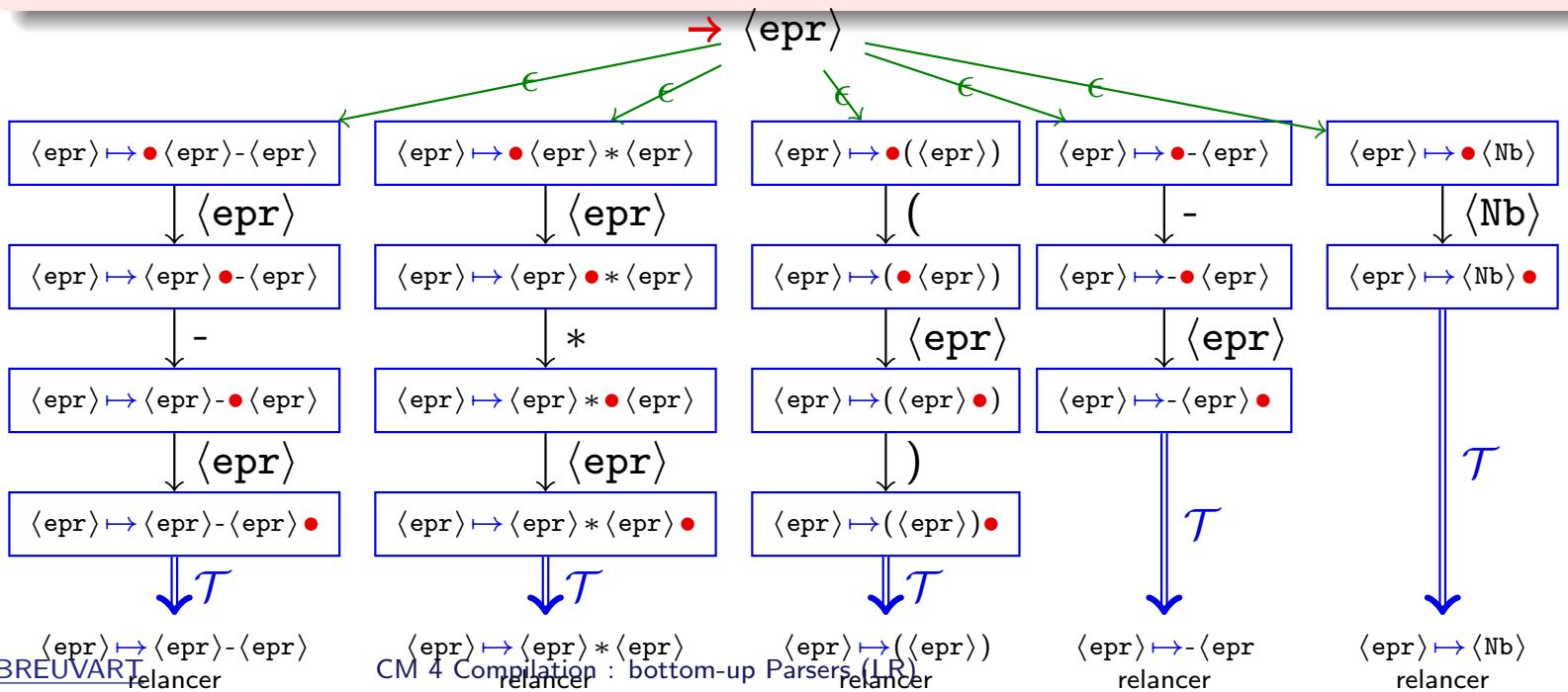
$\rightarrow \langle \texttt{epr} \rangle$



Second attempt

Actions taken on terminal peak are easy to deal with, but how to try a rule where we may need the created NT

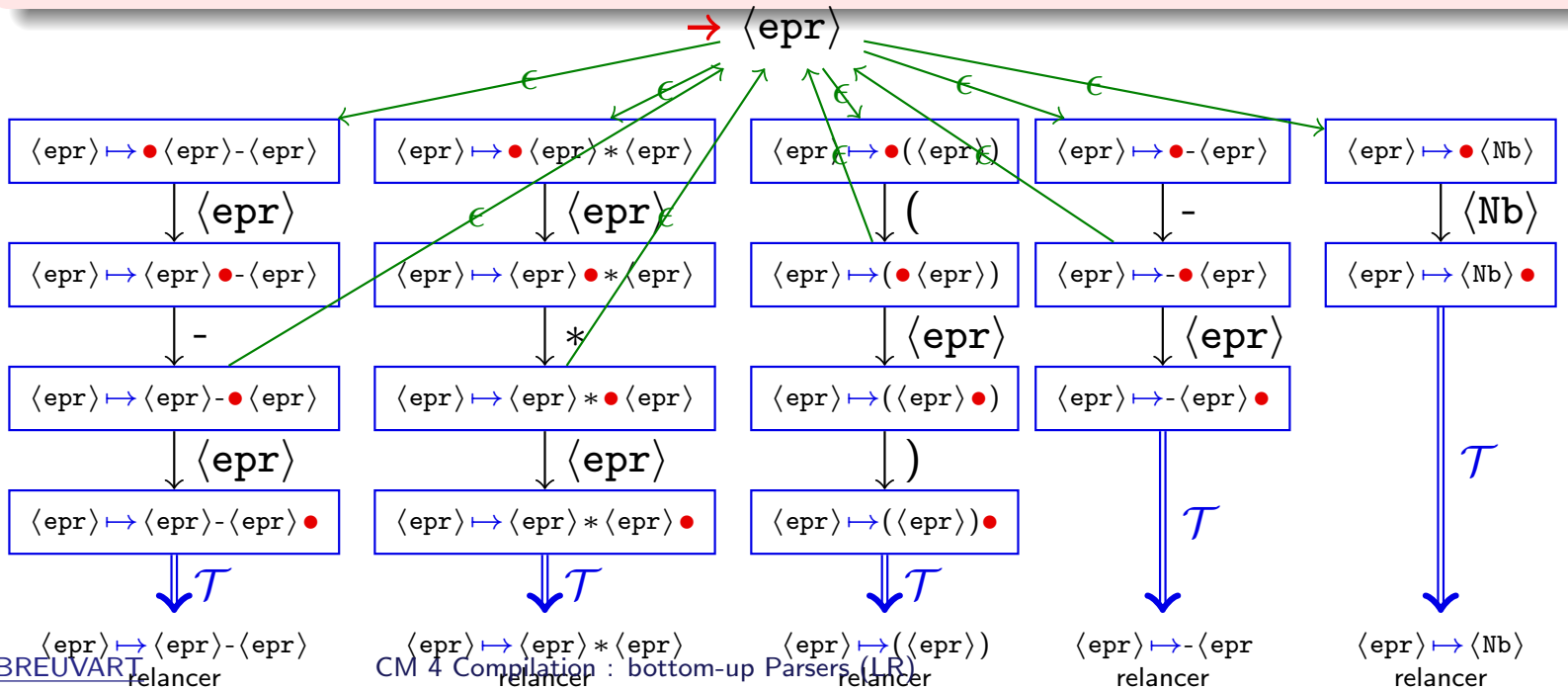
- add a dumy state for each non-terminal N,
  meaning : "I neet to create a N"
- only the pricipal one S is initial,
- add ϵ-transitions toward rules, meaning : "choose a way to reduce it"



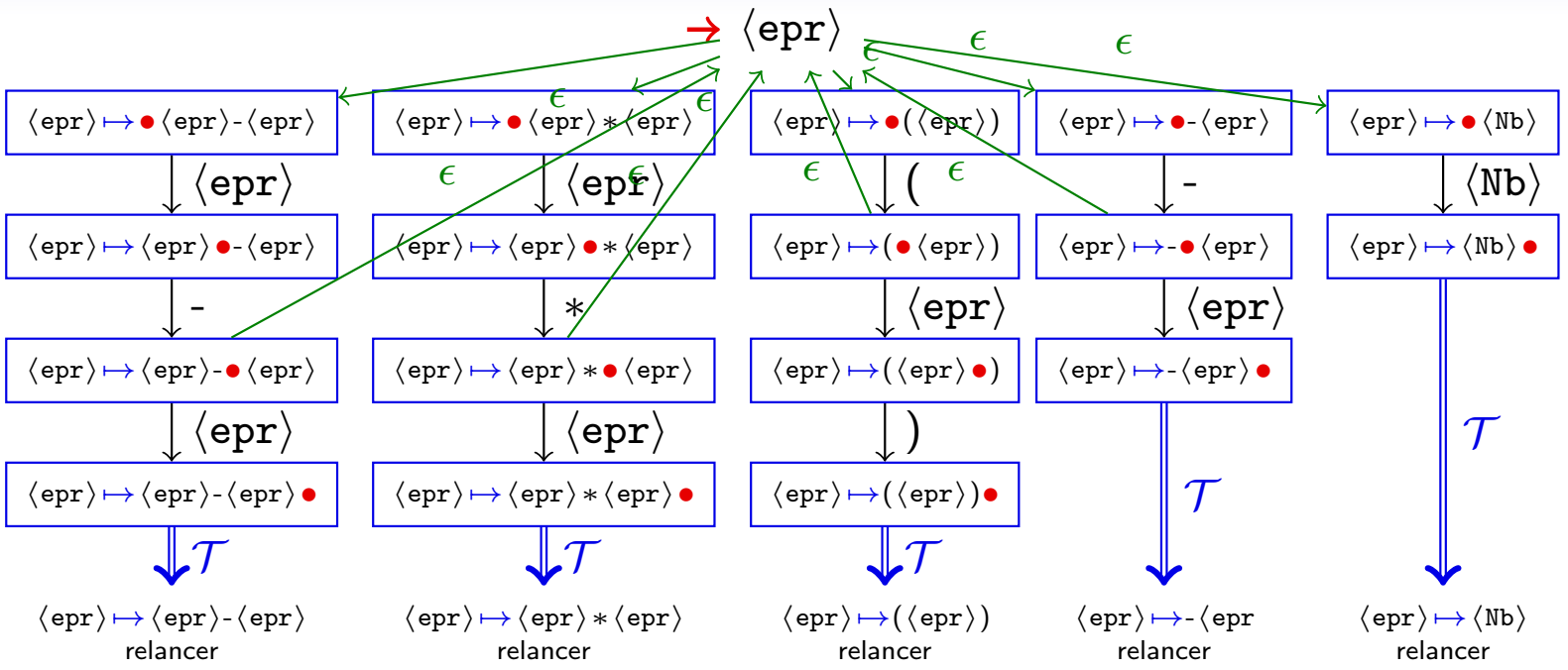
Second attempt

Actions taken on terminal peak are easy to deal with, but how to try a rule where we may need the created NT

- add a dumy state for each non-terminal N,
  meaning : "I neet to create a N"
- only the pricipal one S is initial,
- add ϵ-transitions toward rules, meaning : "choose a way to reduce it"
- for each rules of the form $M \mapsto w_1 \bullet N w_2$, add an ϵ-transition toward N
  meaning : "a N is expected here, we may create it"



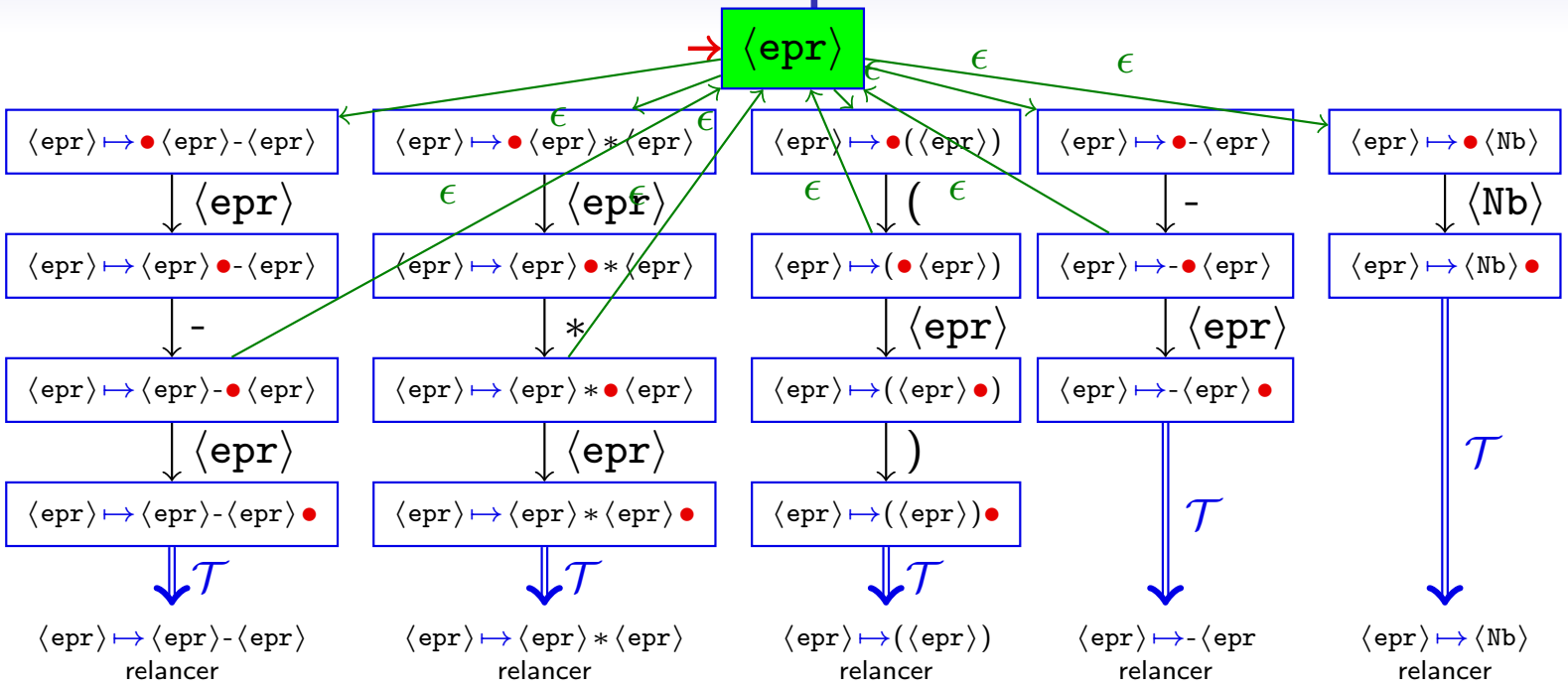
Second attempt in motion

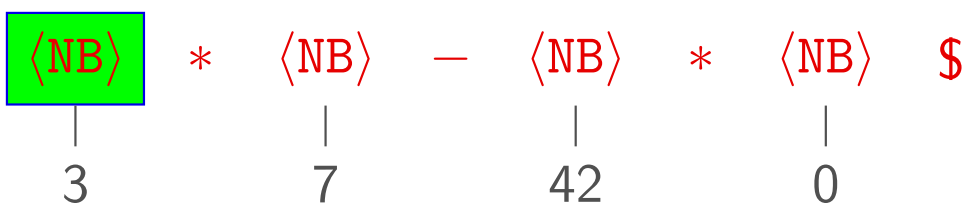


⟨NB⟩ ∗ ⟨NB⟩ − ⟨NB⟩ ∗ ⟨NB⟩ $

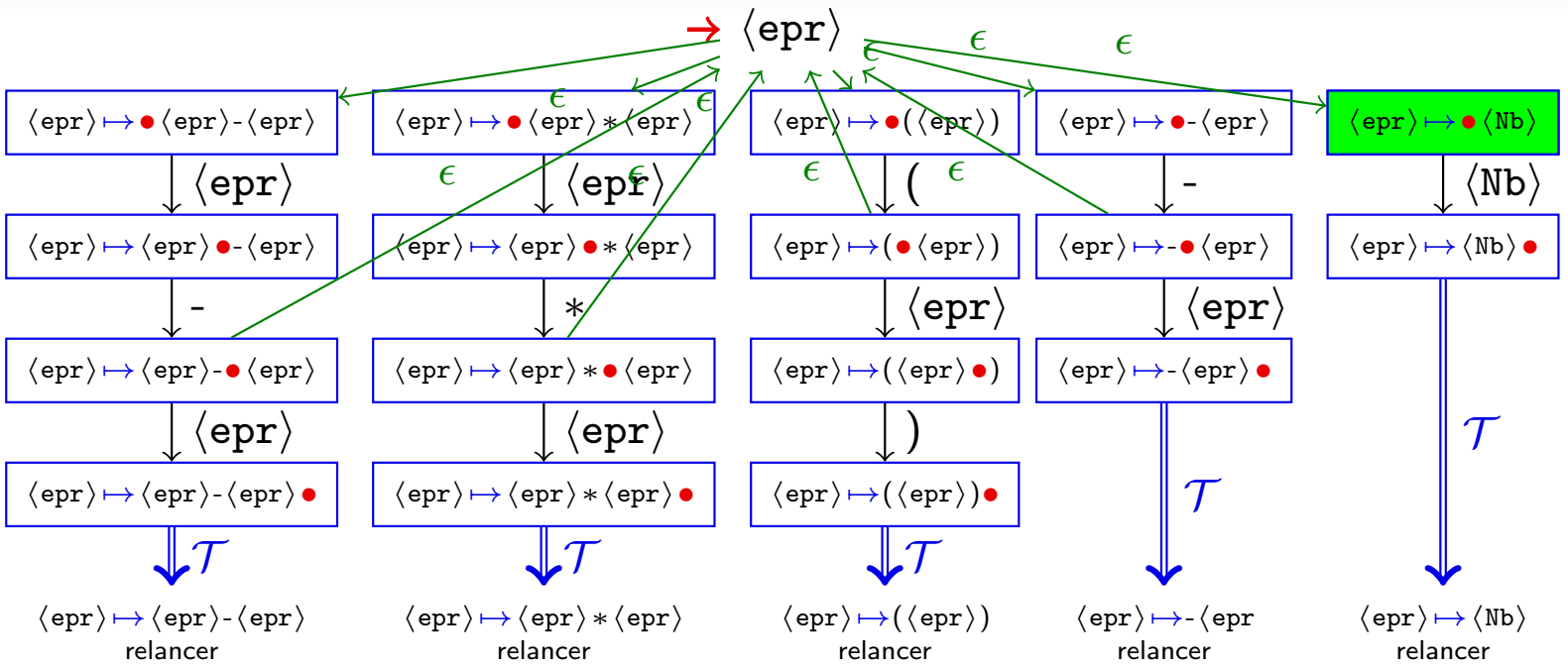
3 7 42 0

Second attempt in motion

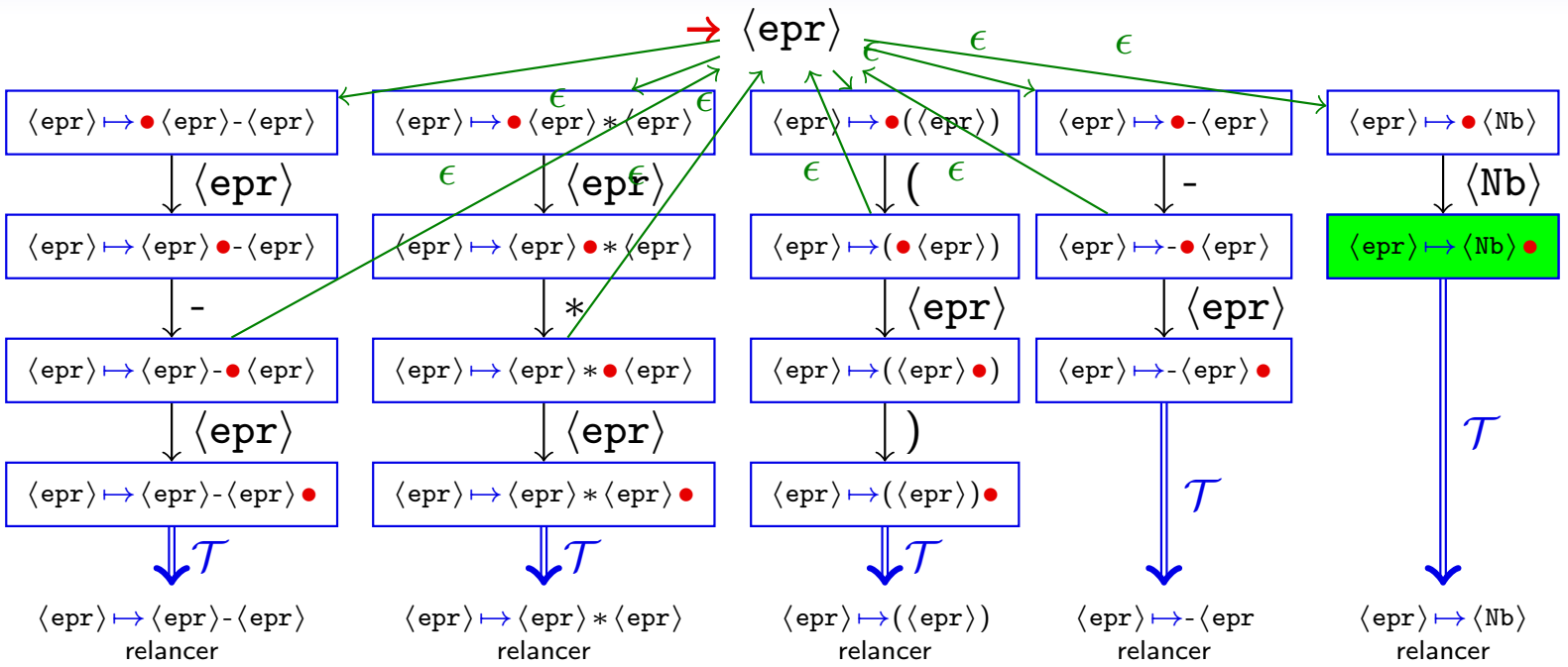


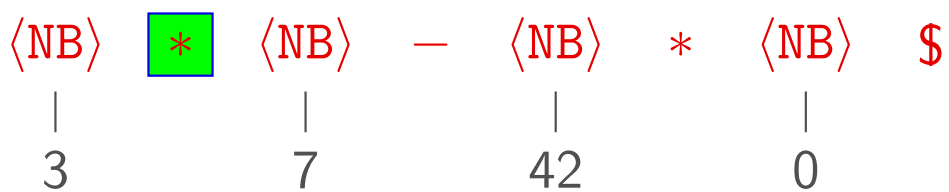


Second attempt in motion

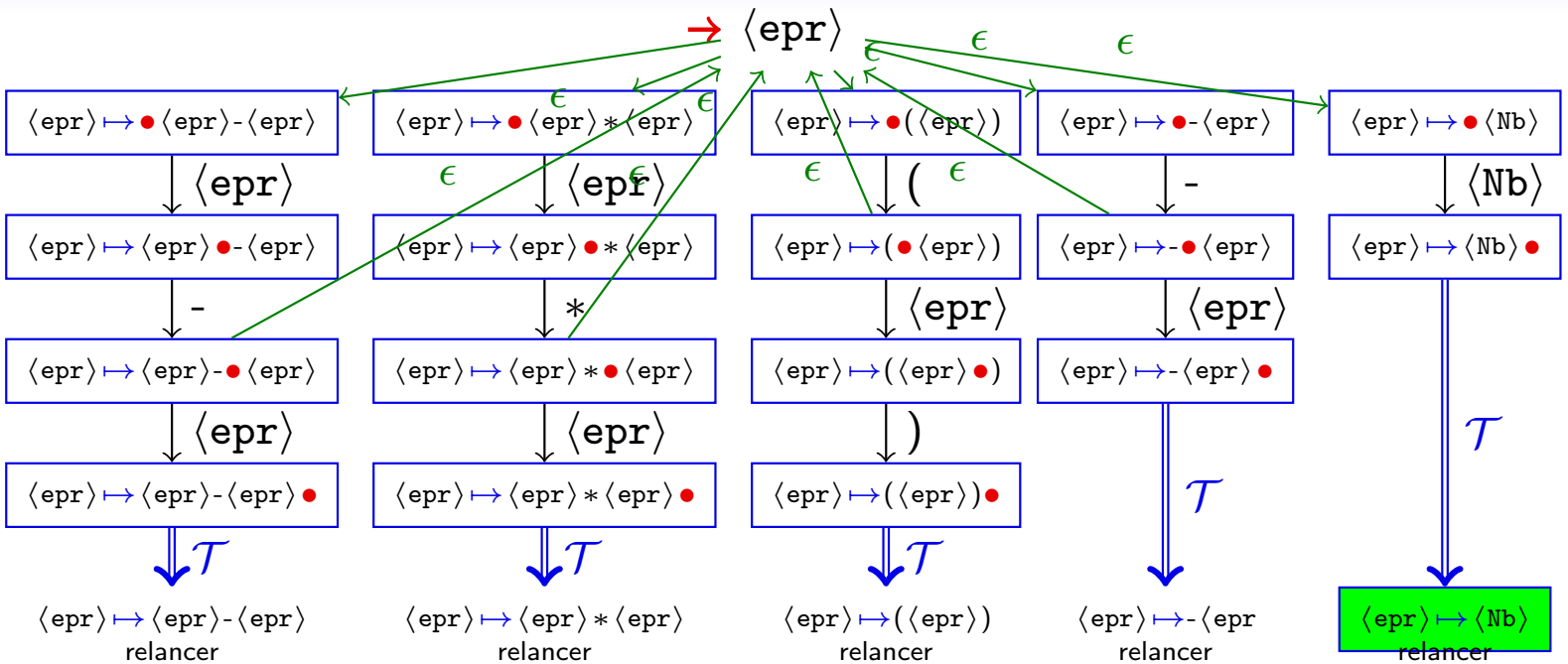


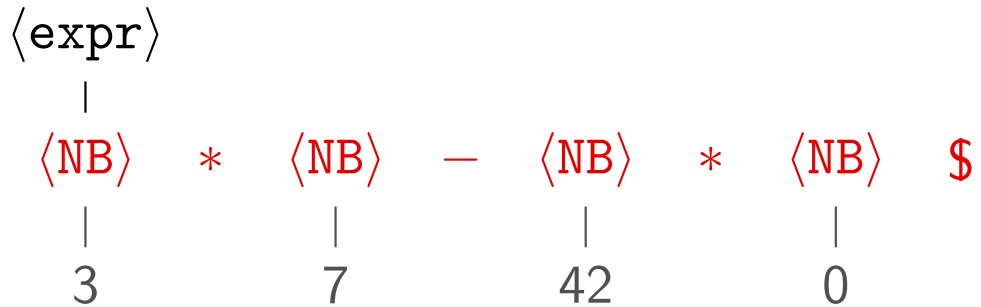
Second attempt in motion



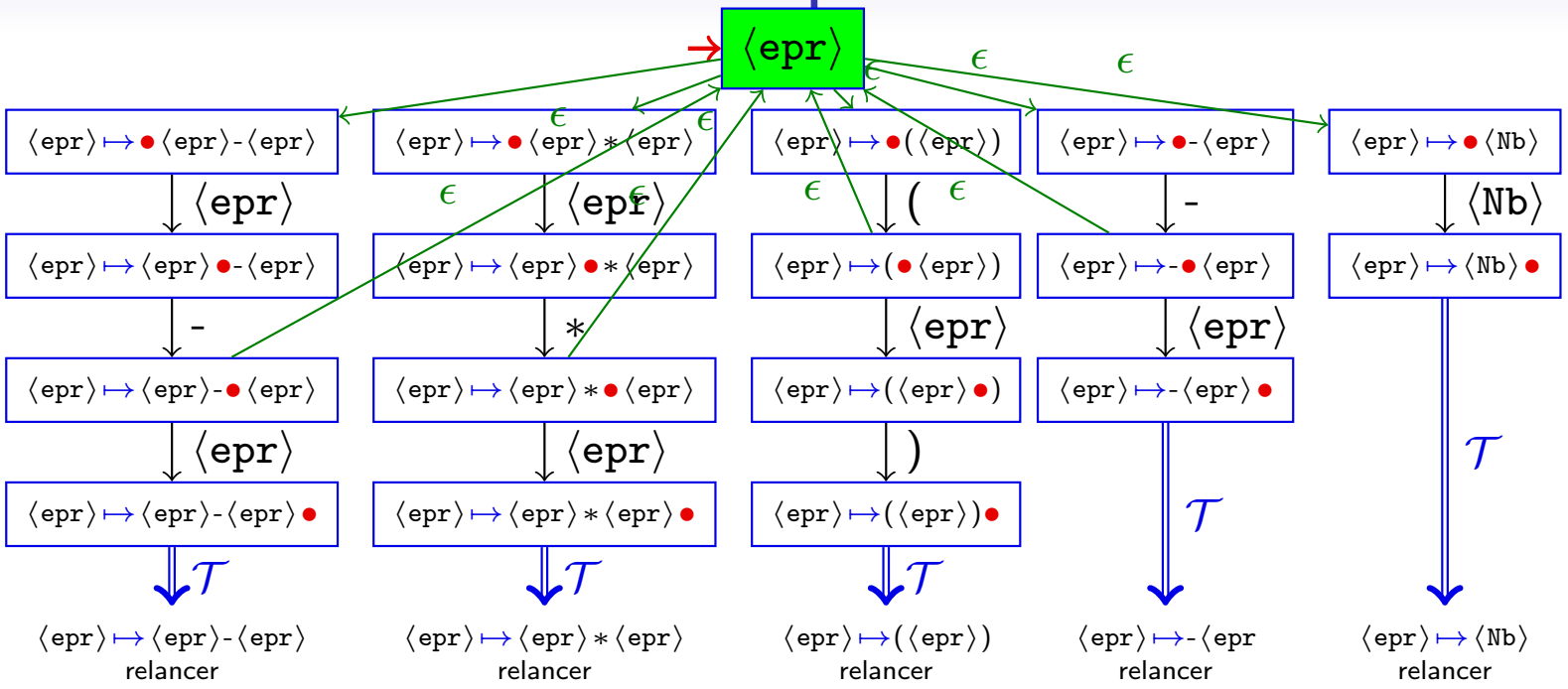


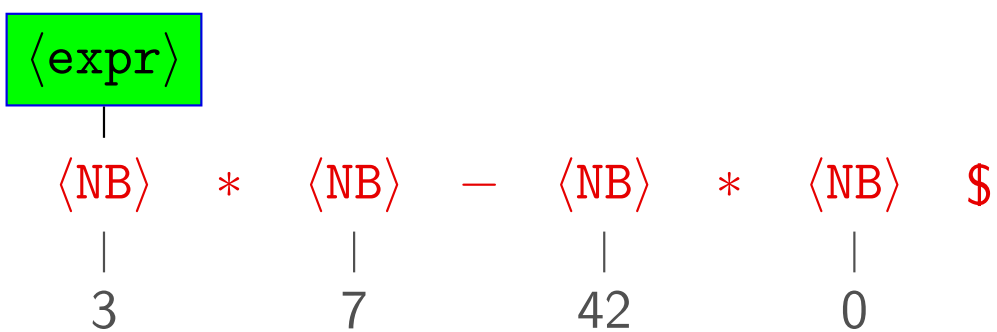
Second attempt in motion



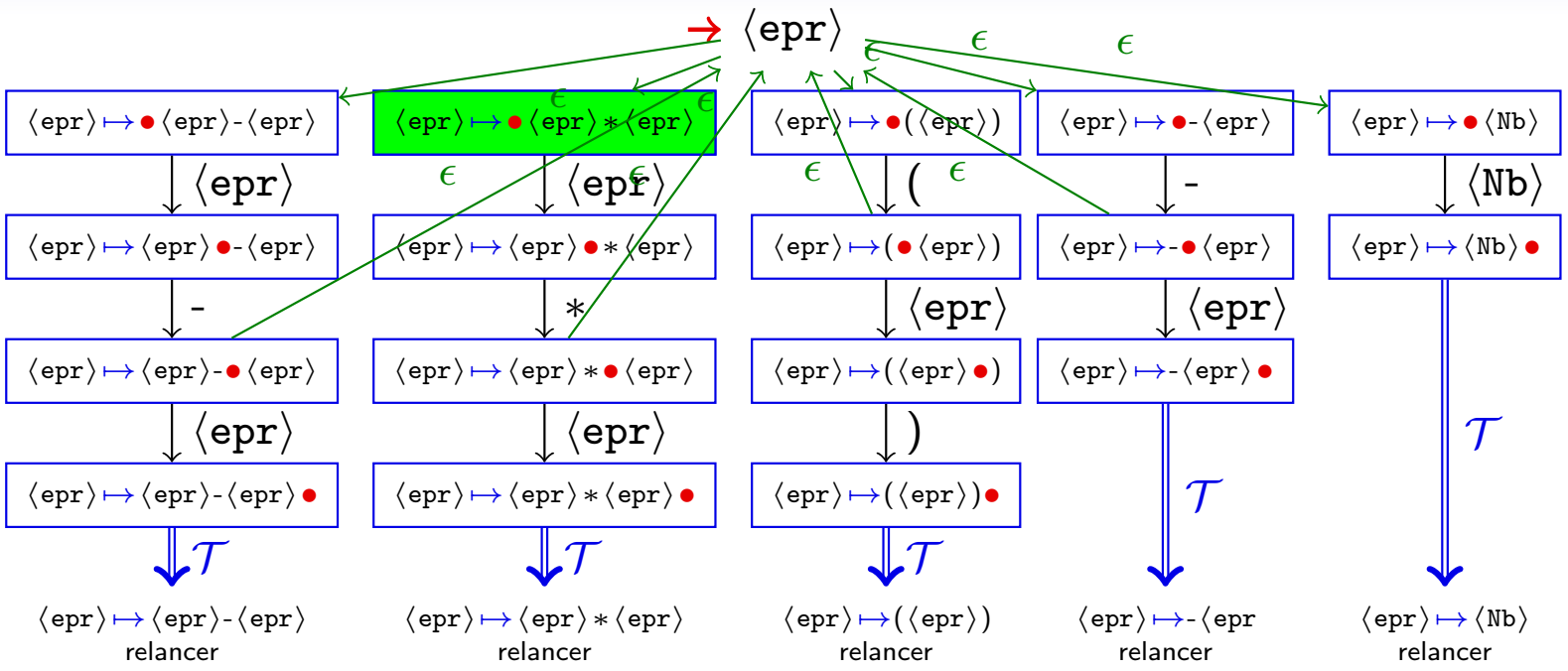


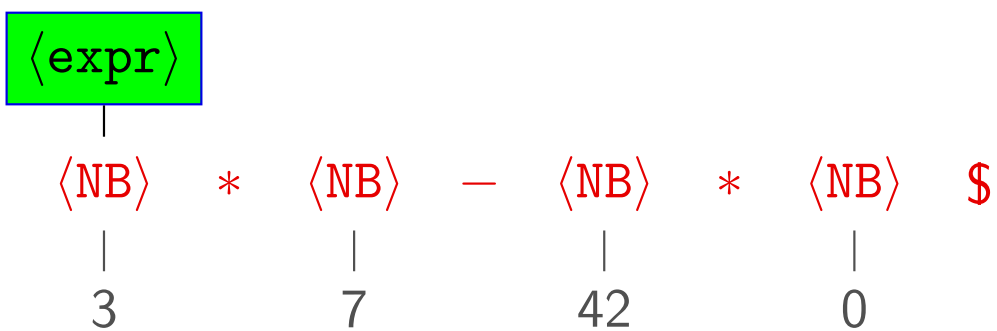
Second attempt in motion



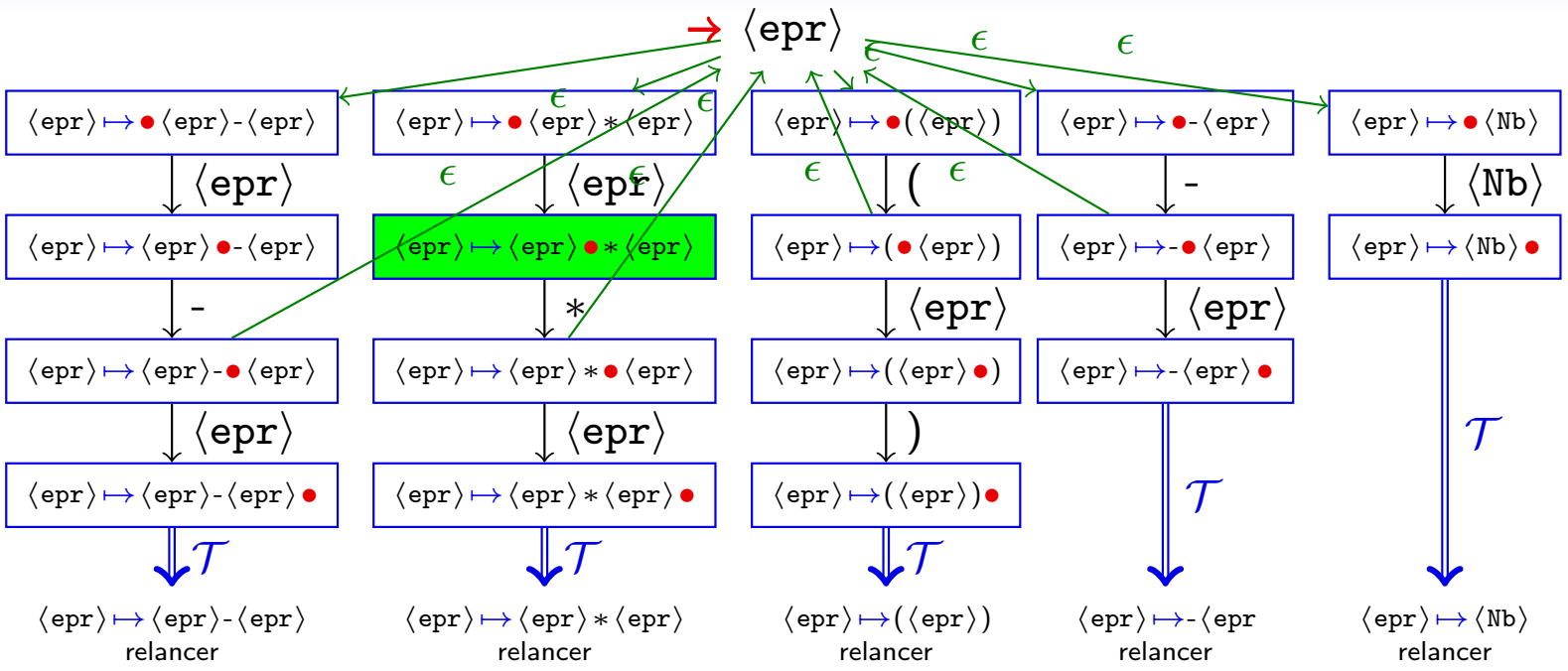


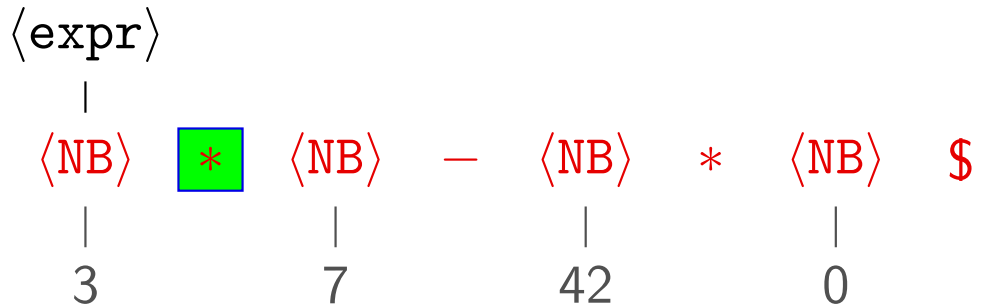
Second attempt in motion



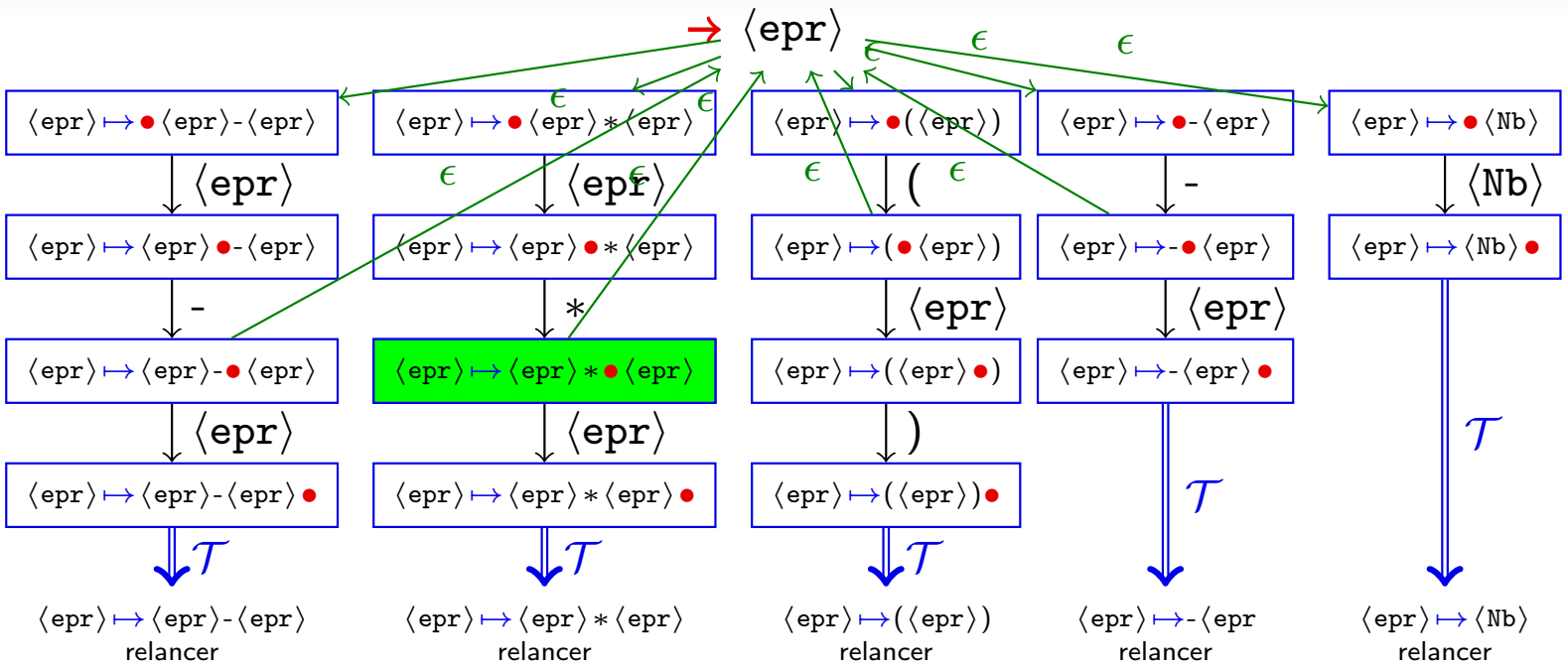


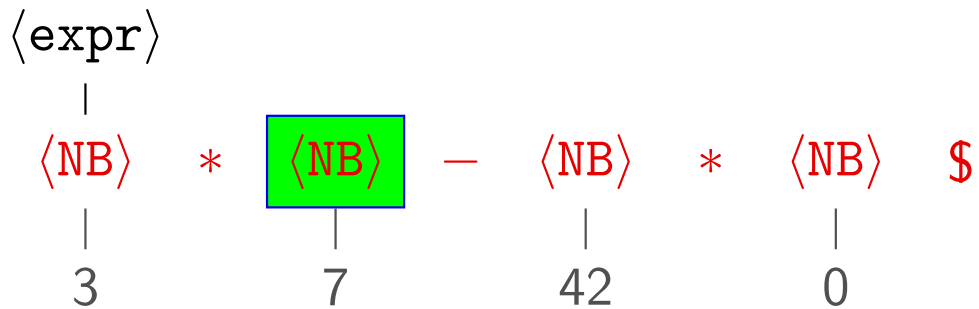
Second attempt in motion



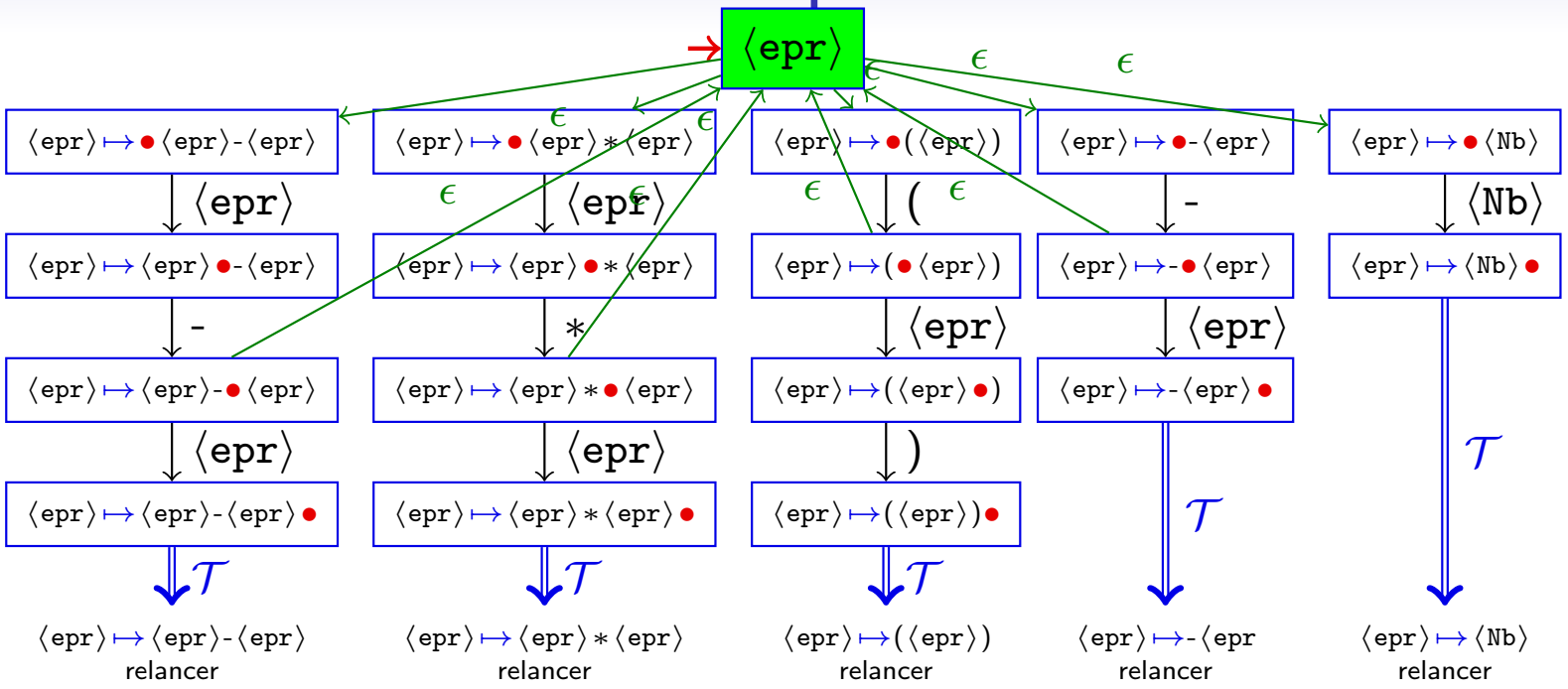


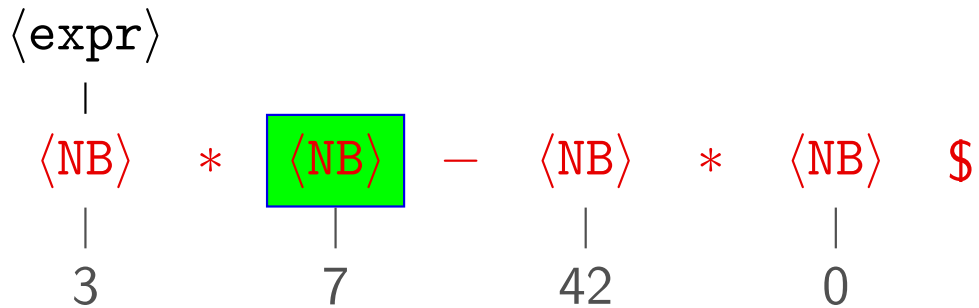
Second attempt in motion



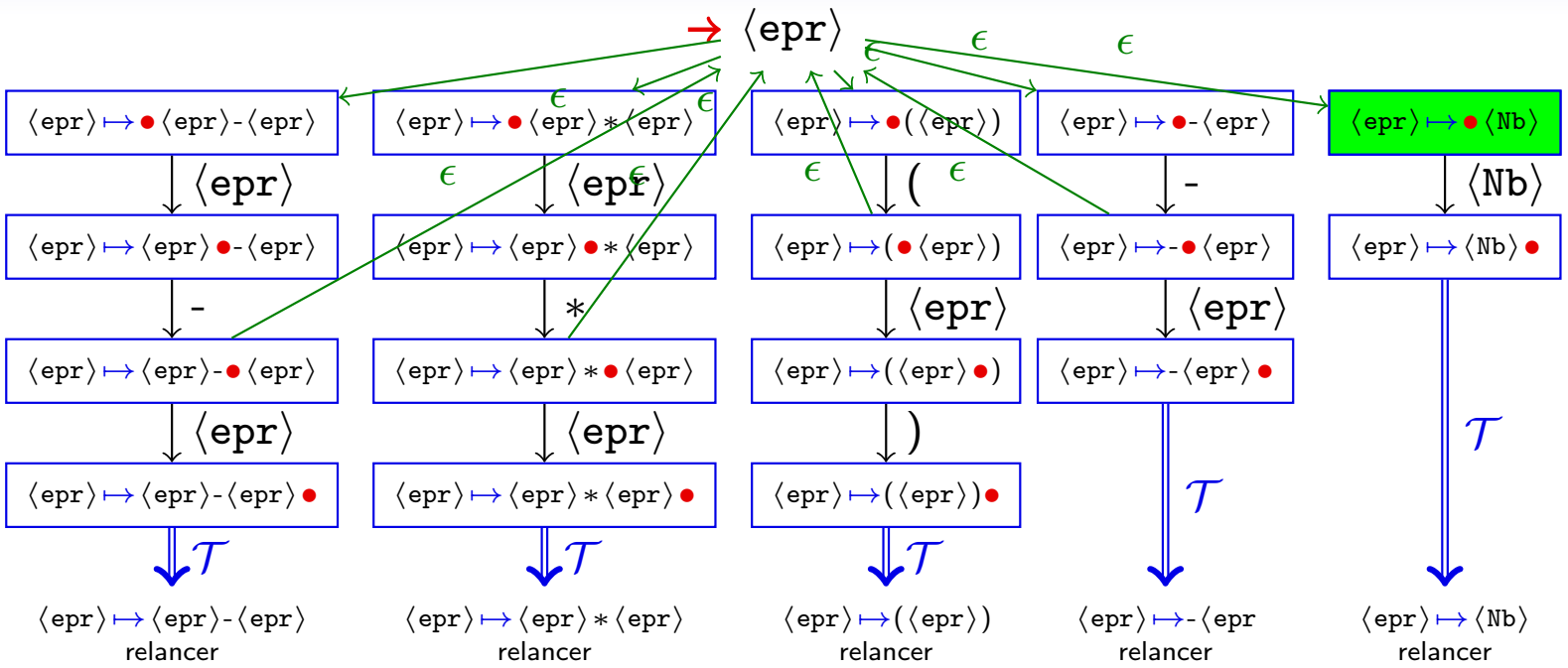


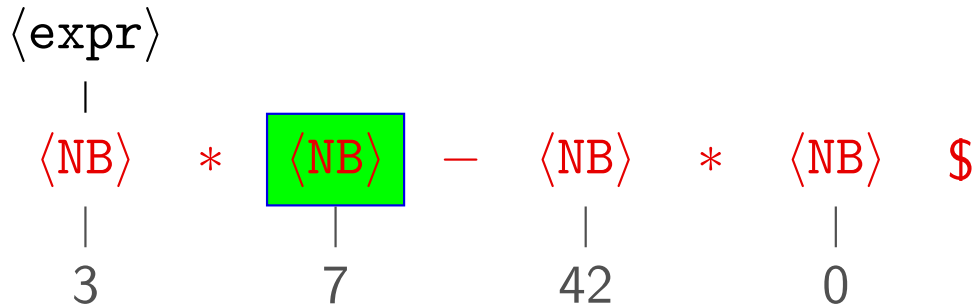
Second attempt in motion



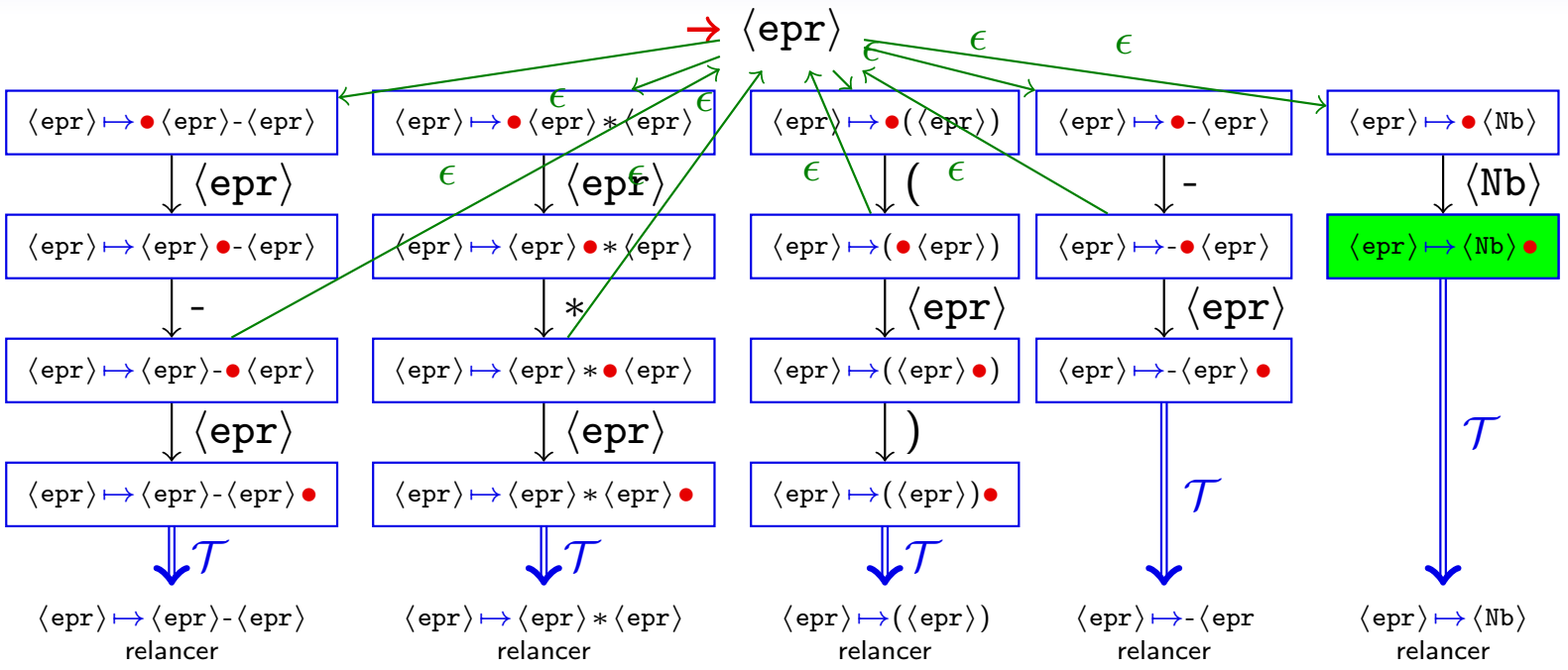


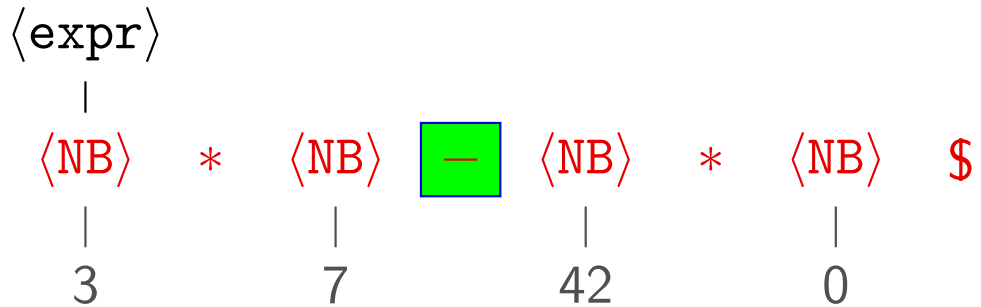
Second attempt in motion



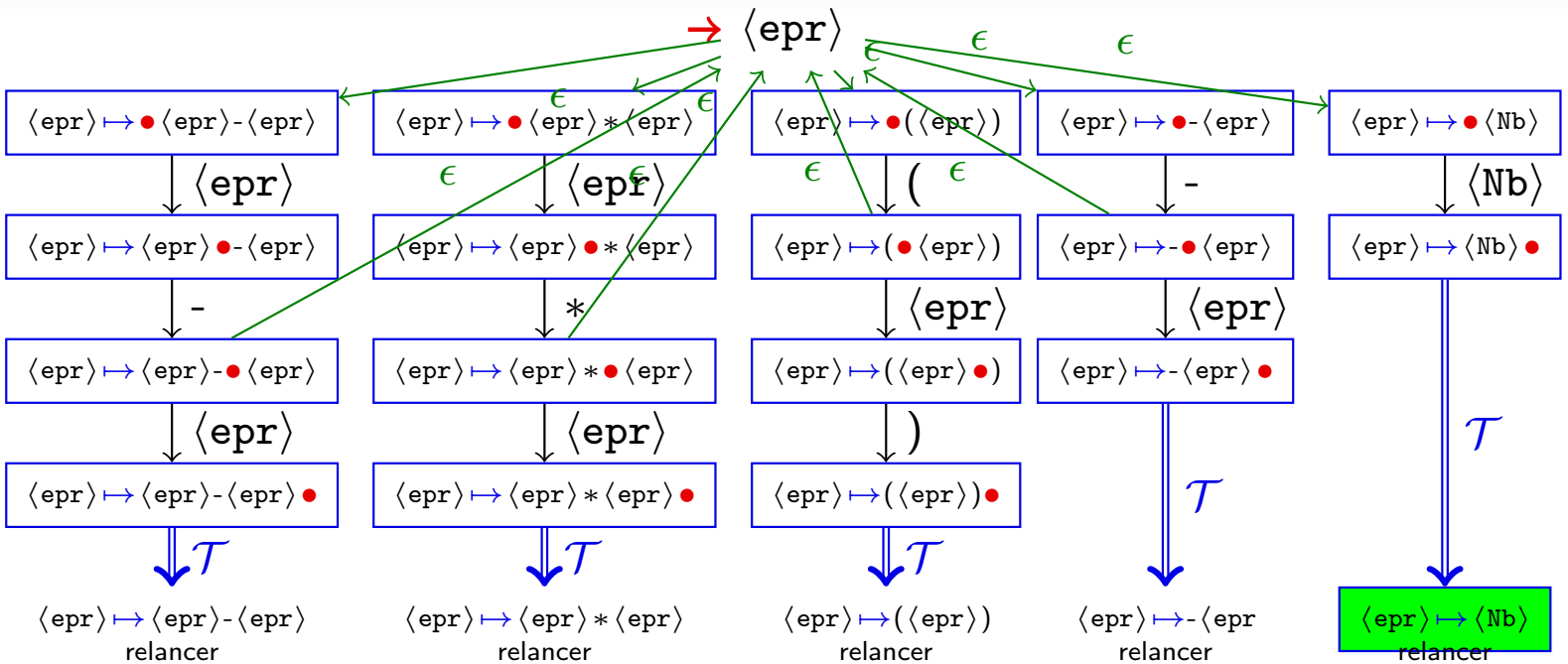


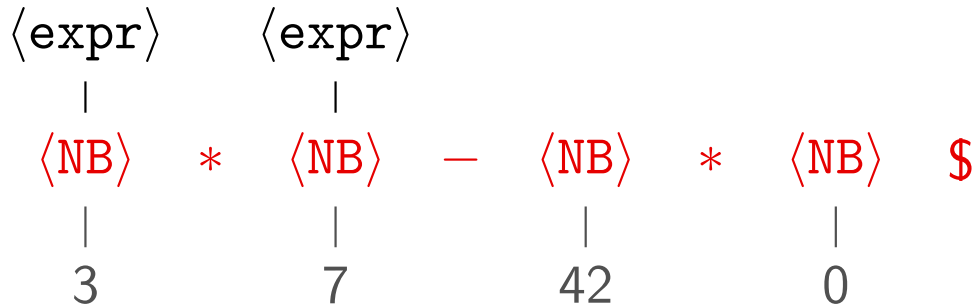
Second attempt in motion



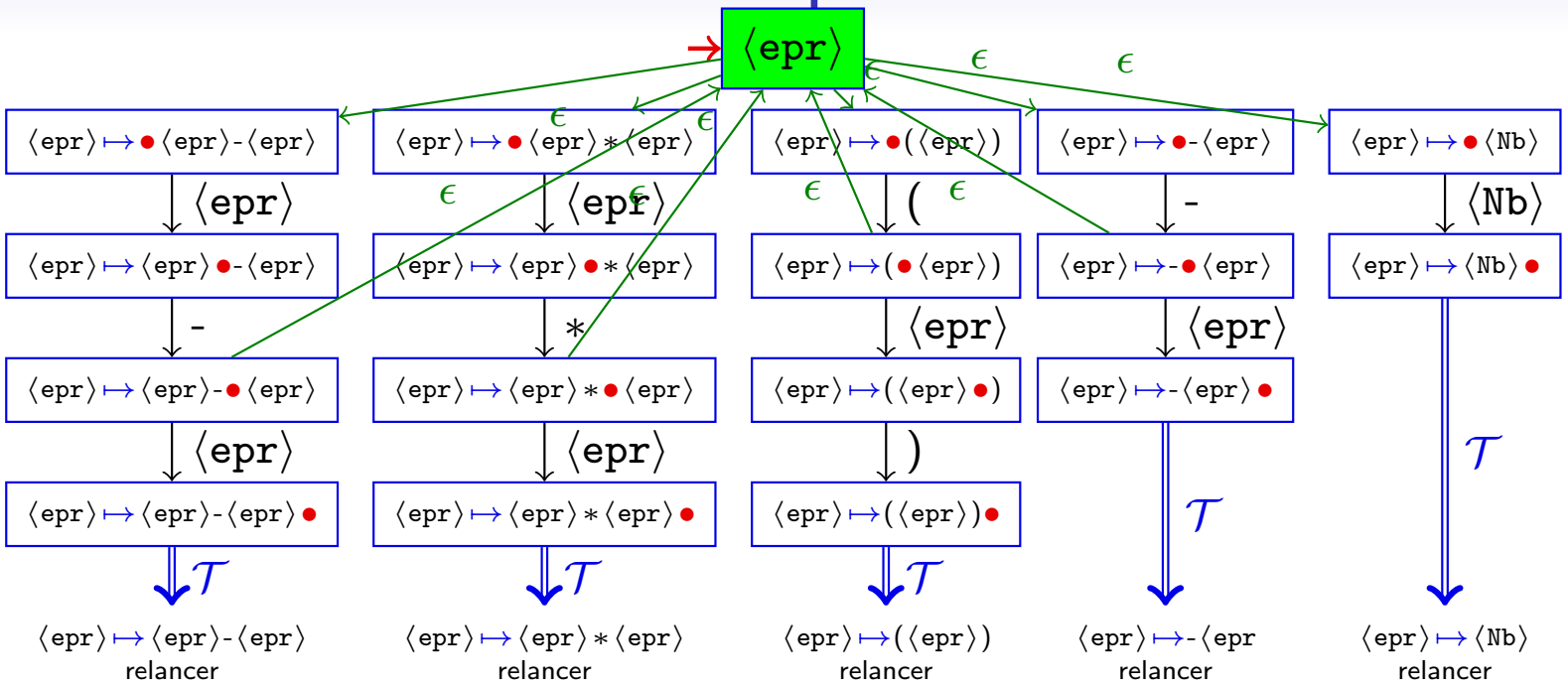


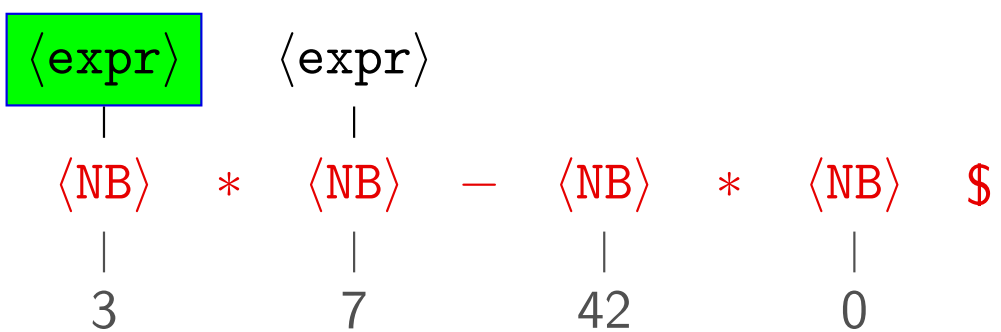
Second attempt in motion



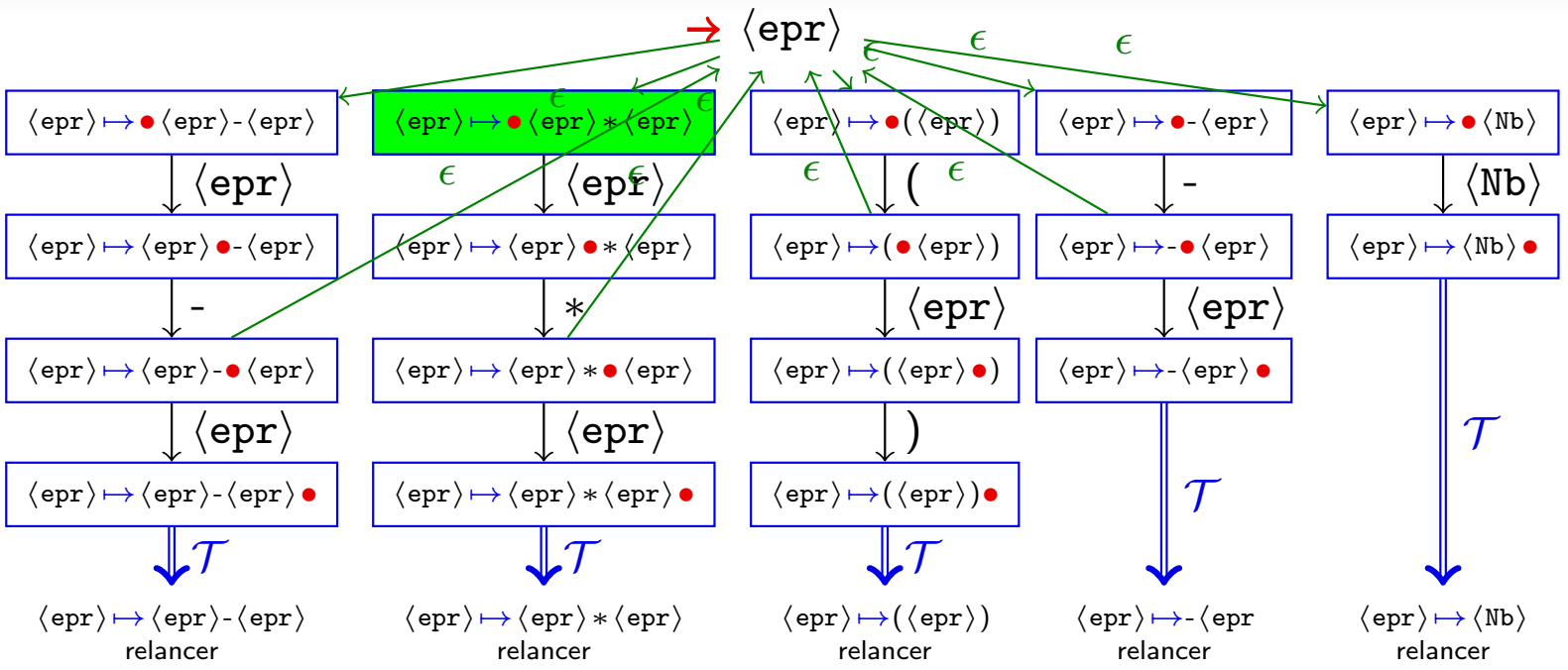


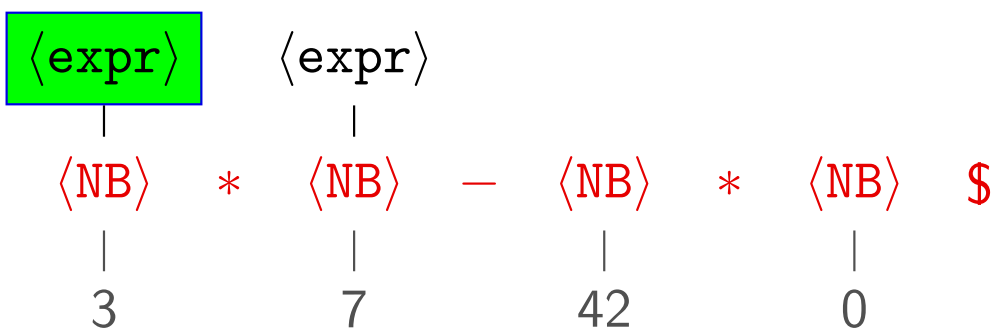
Second attempt in motion



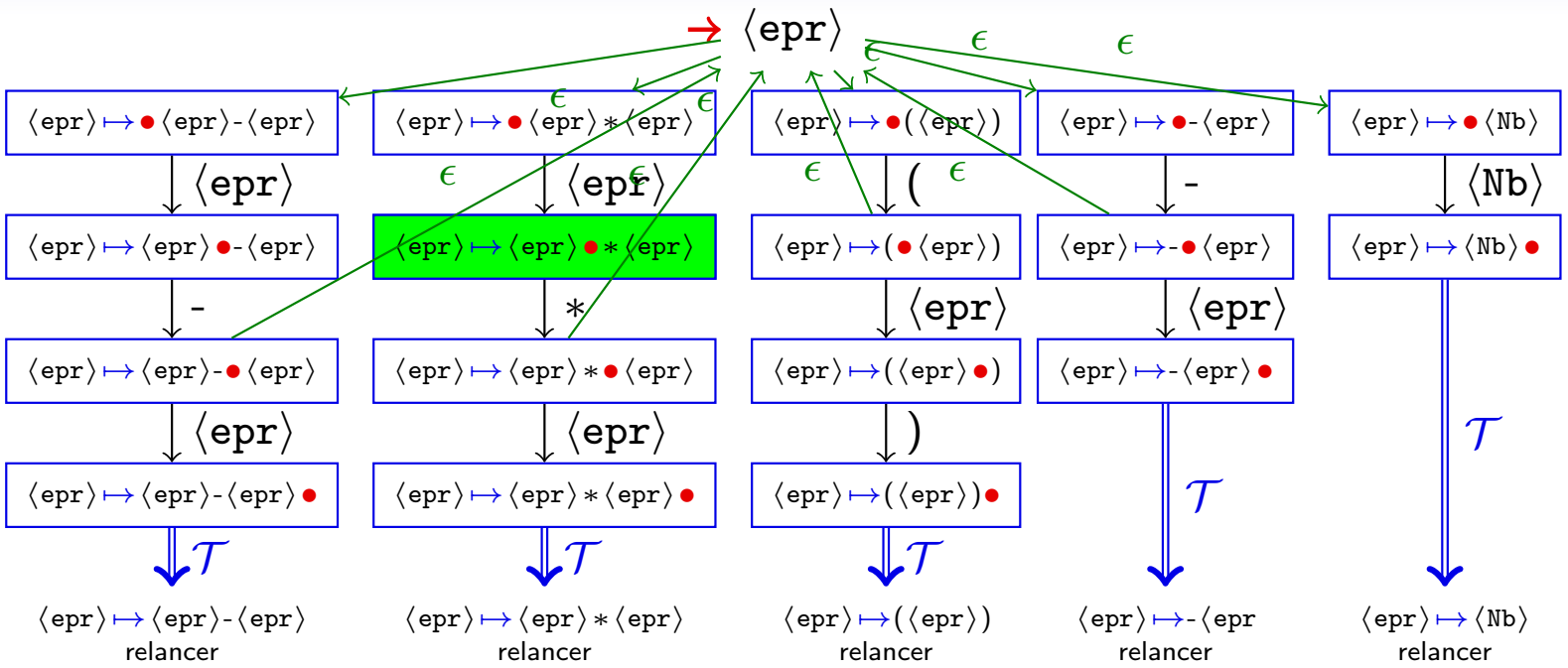


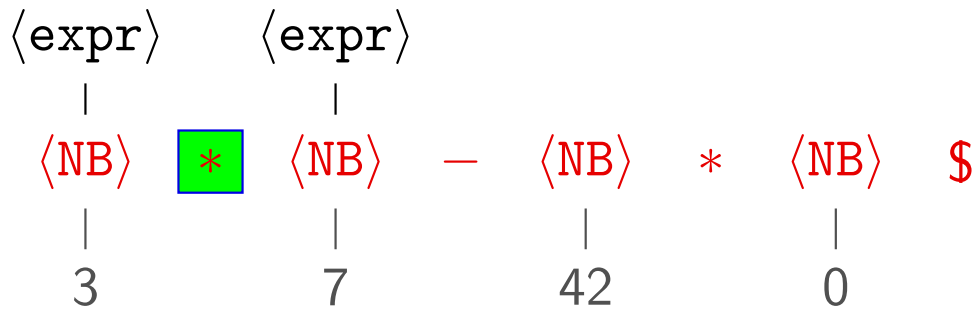
Second attempt in motion



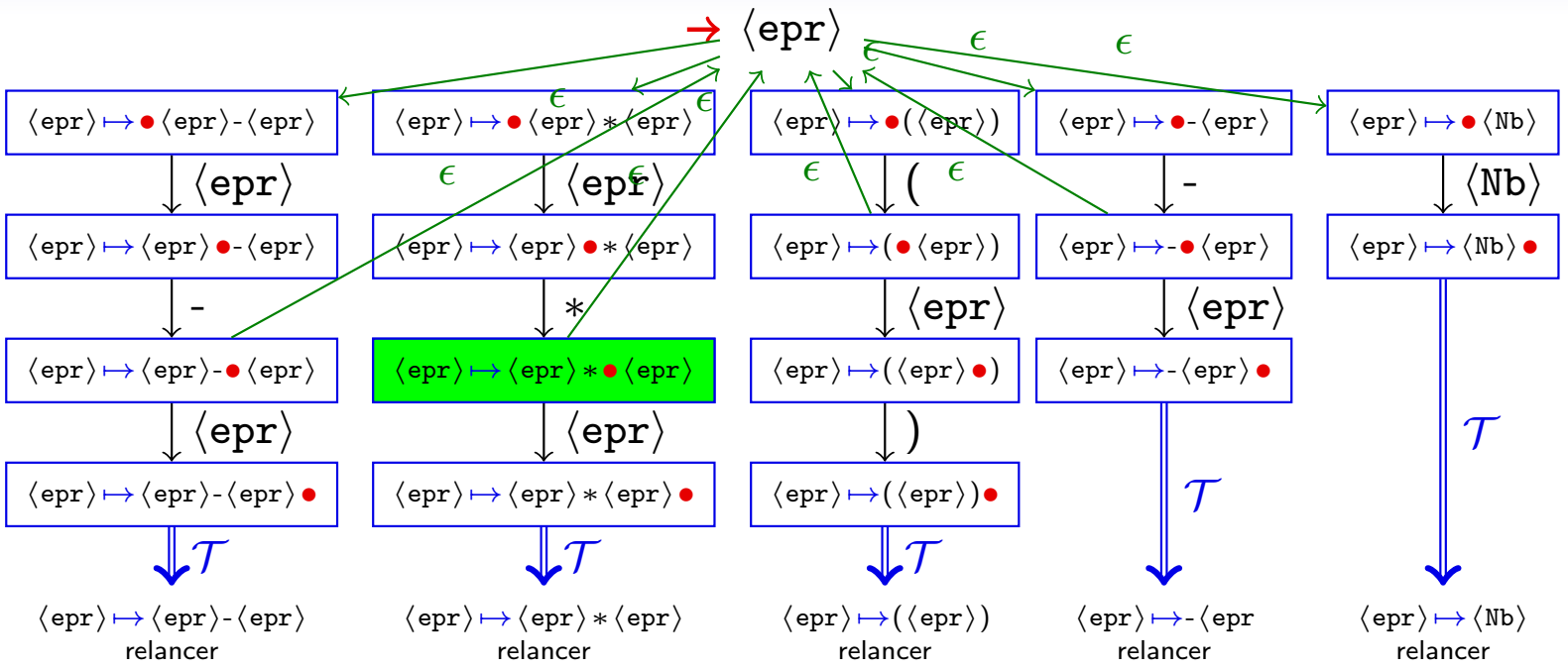


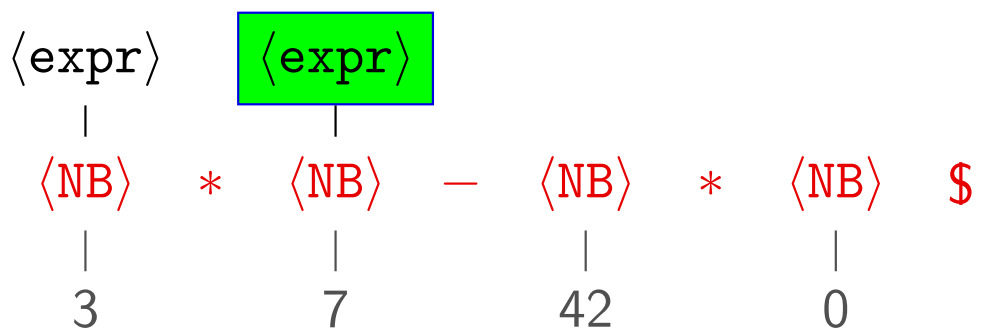
Second attempt in motion



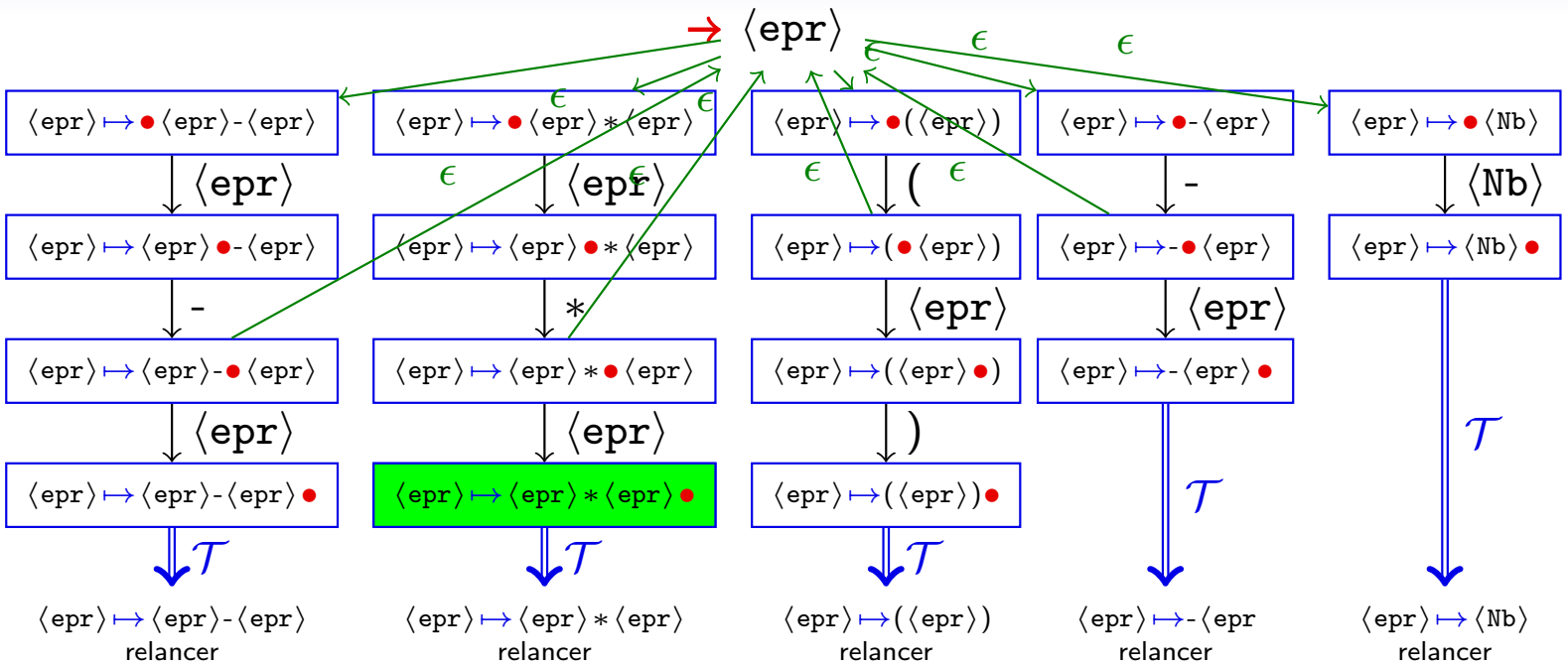


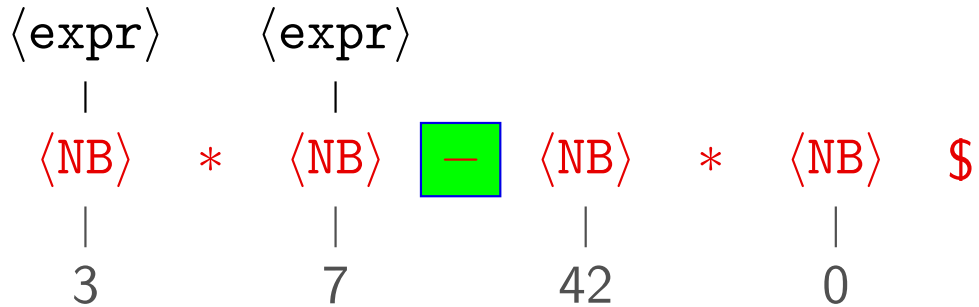
Second attempt in motion



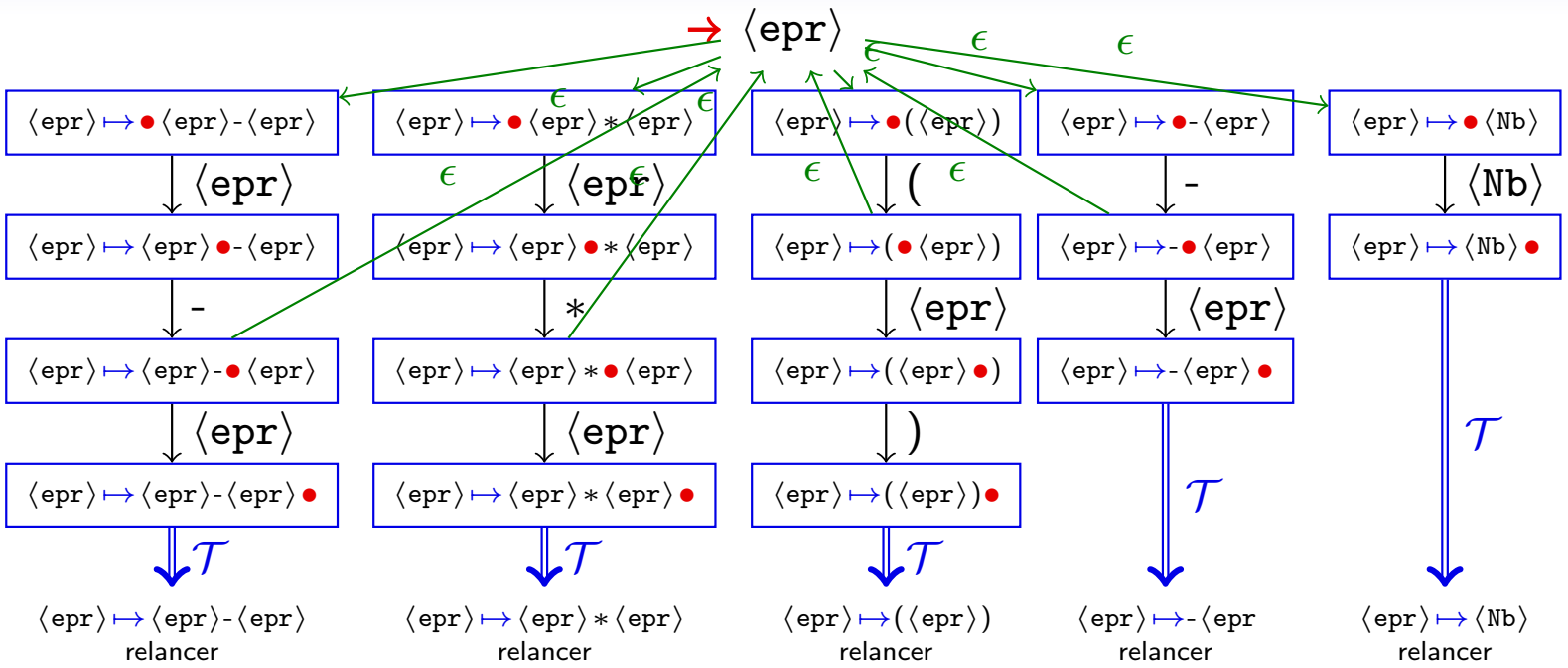


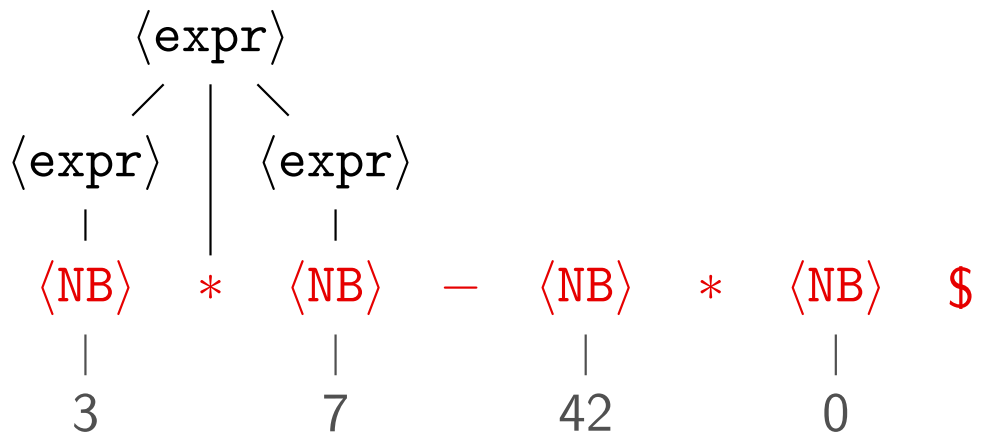
Second attempt in motion



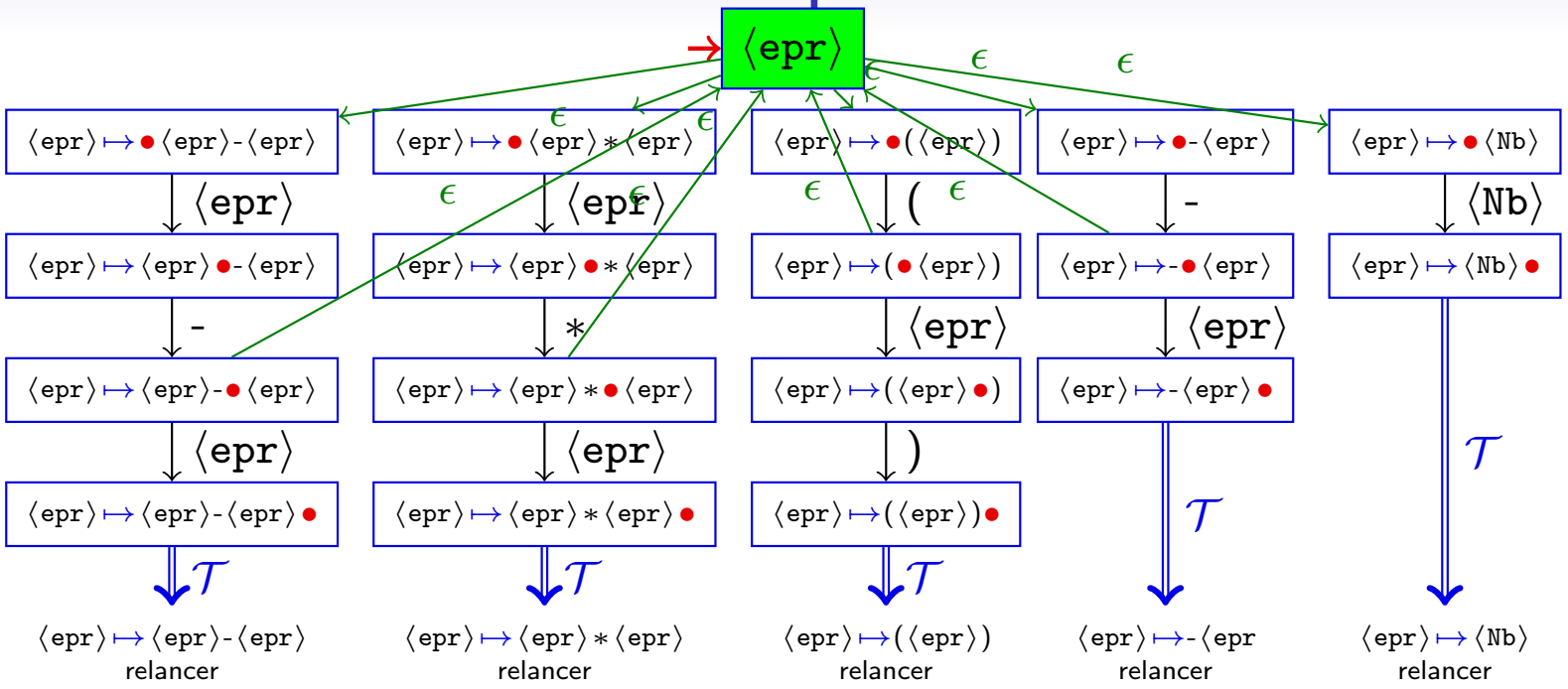


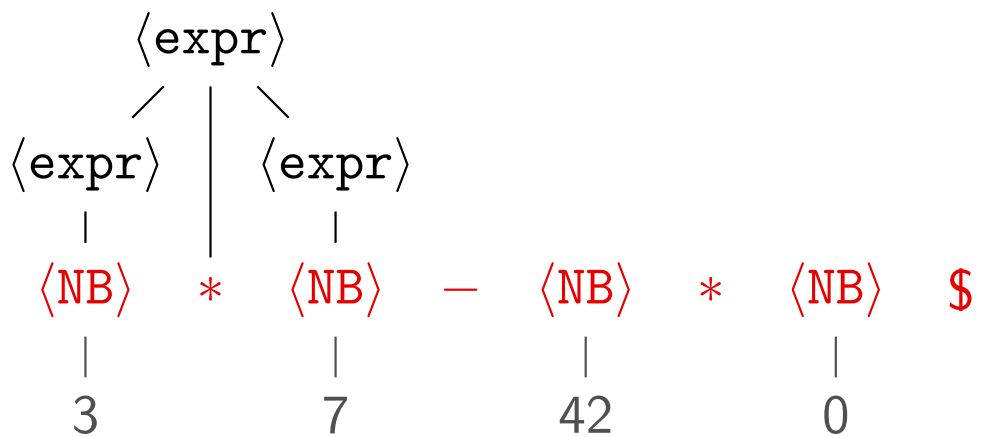
Second attempt in motion



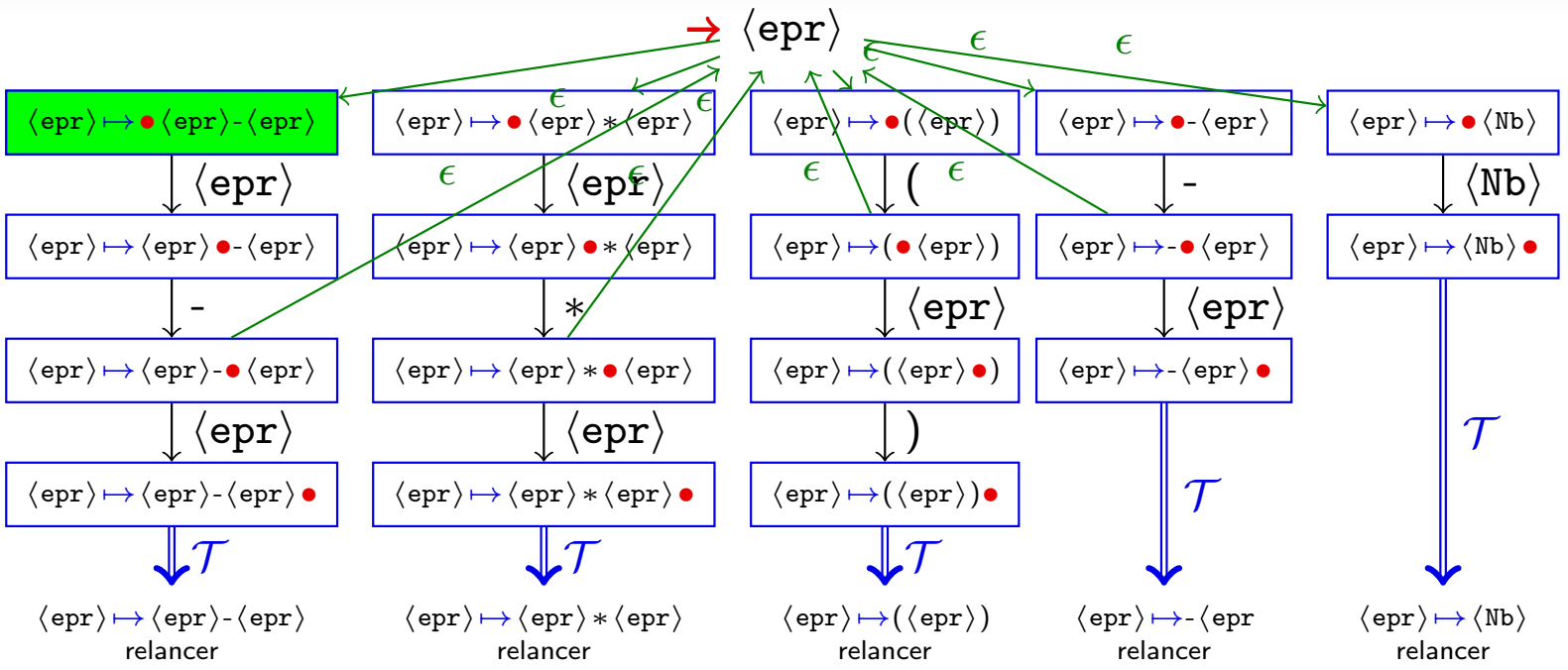


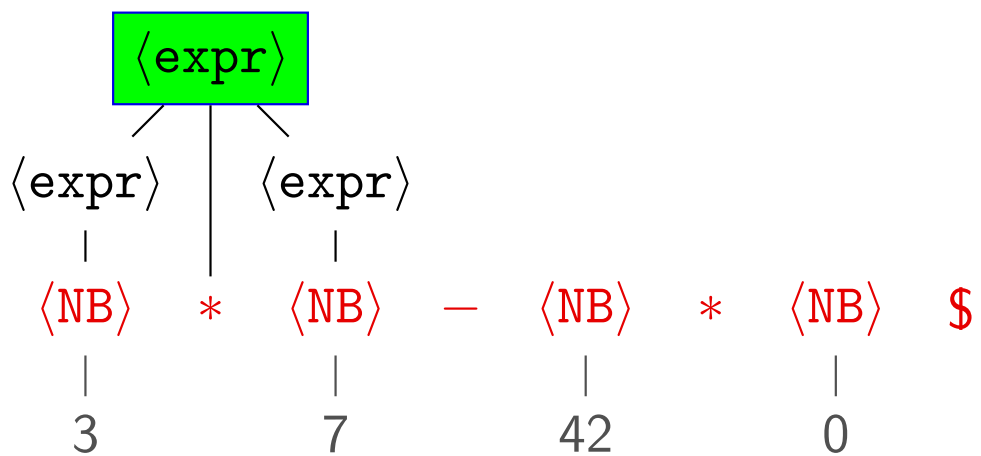
Second attempt in motion



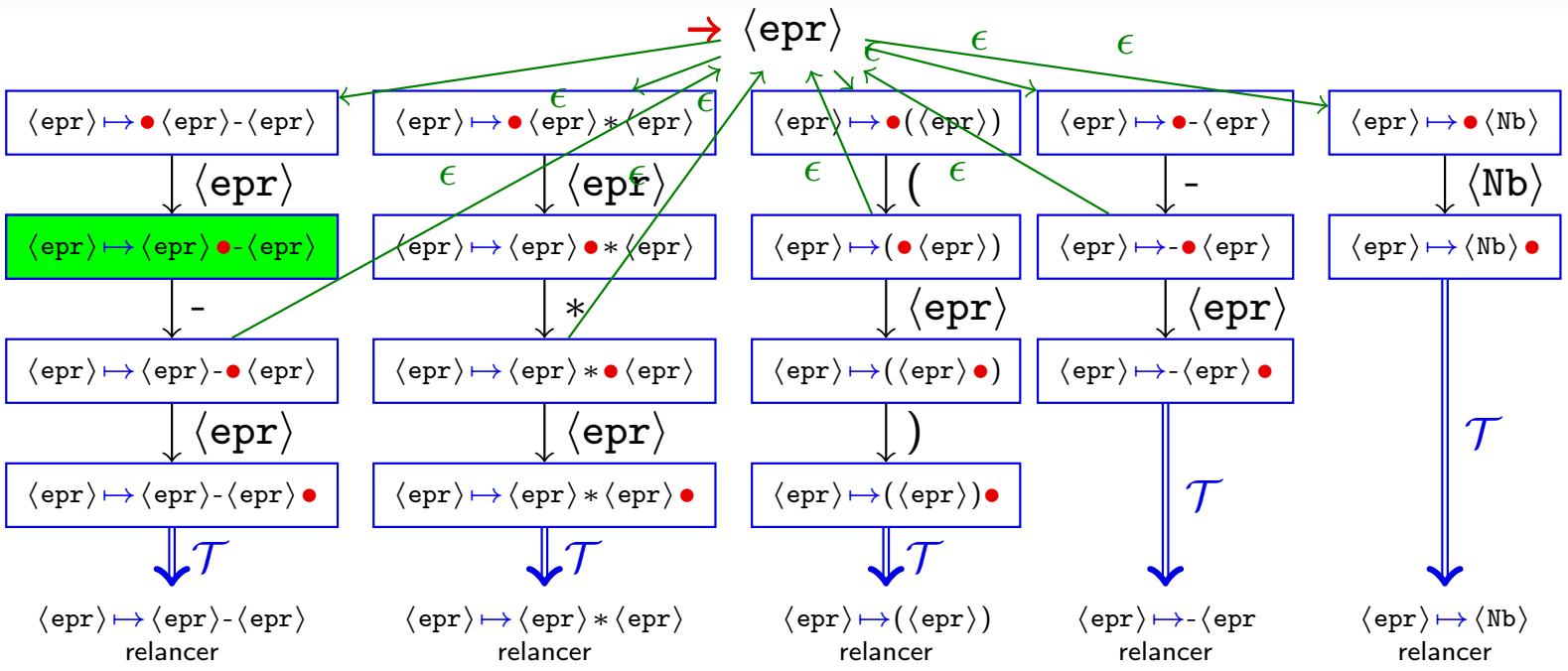


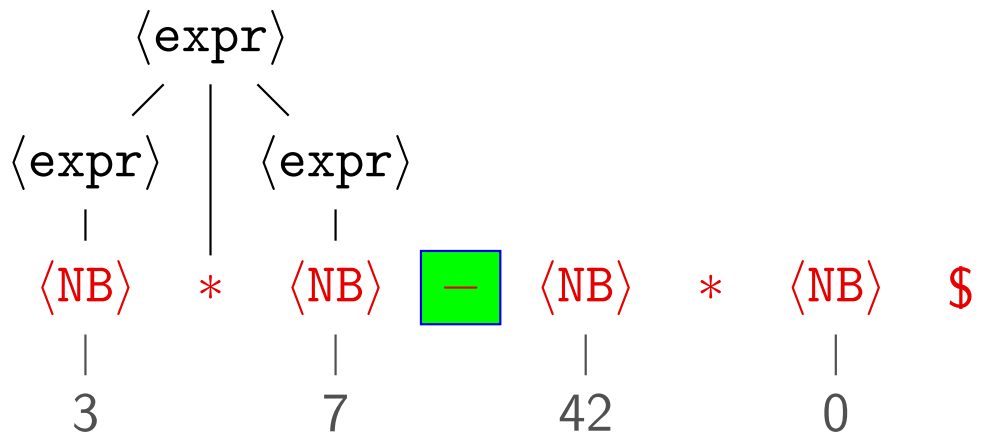
Second attempt in motion



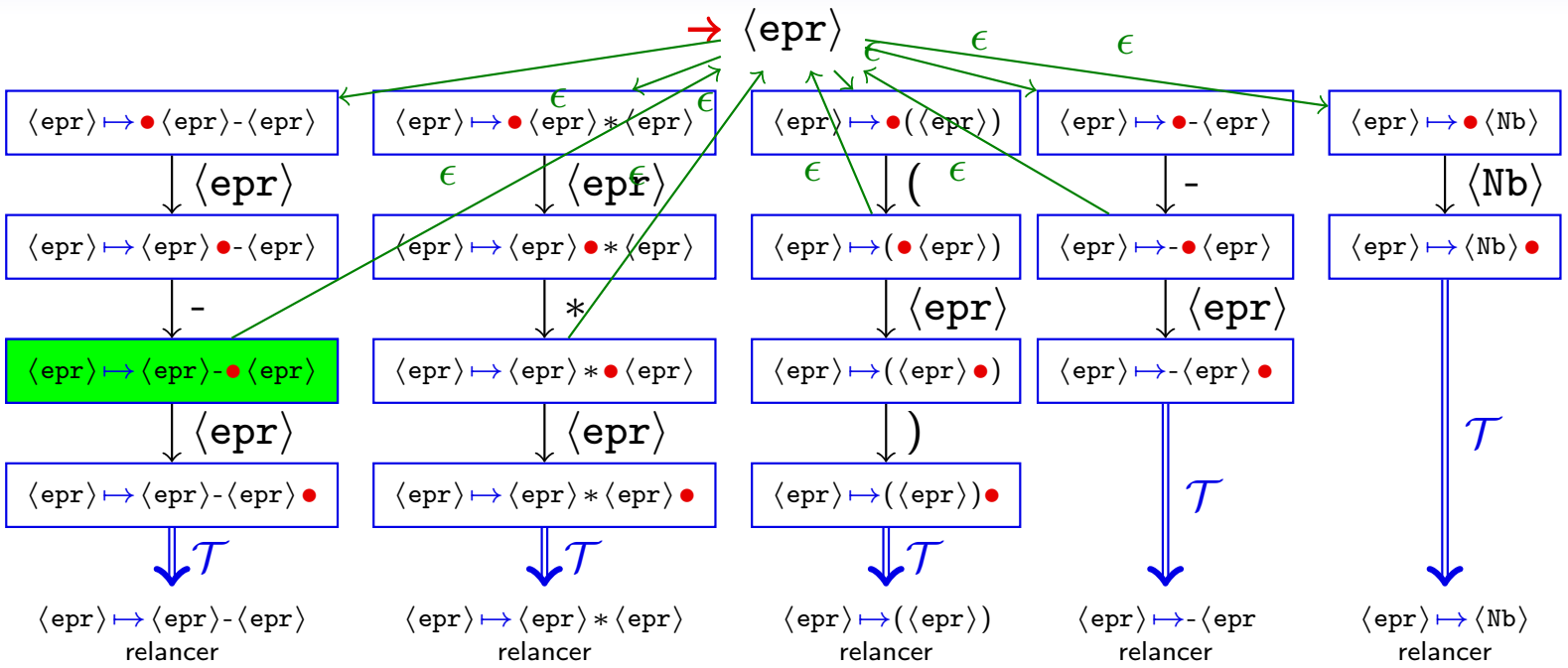


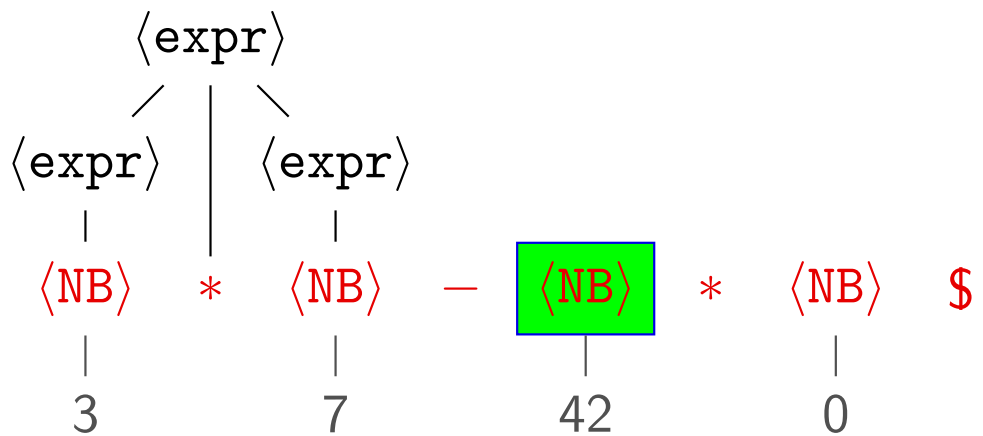
Second attempt in motion



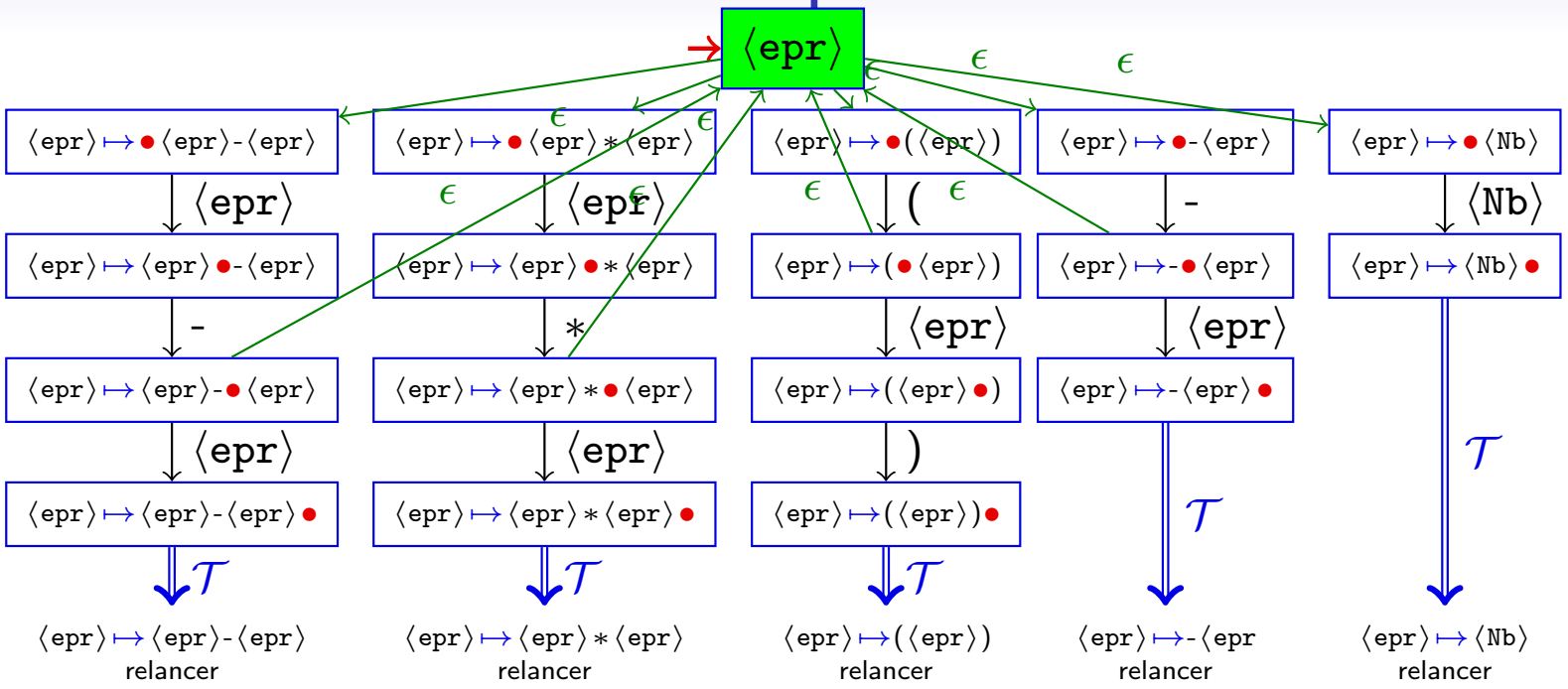


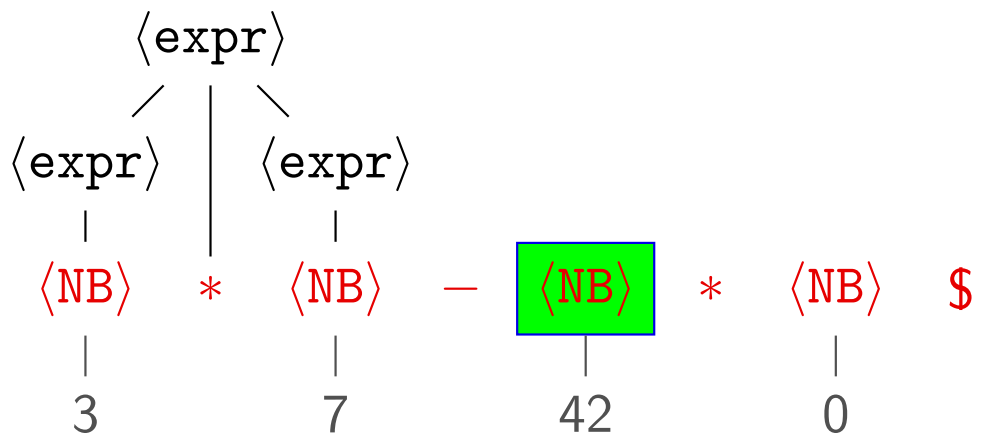
Second attempt in motion



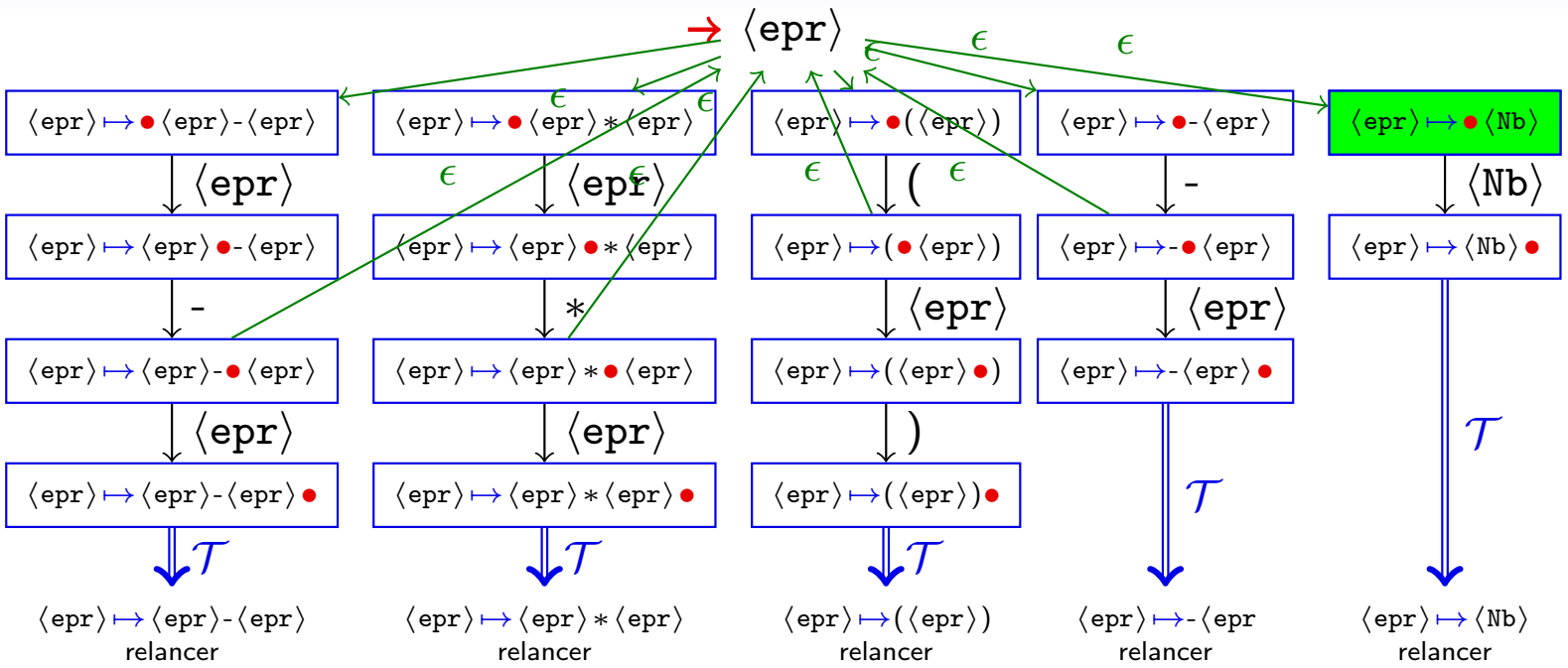


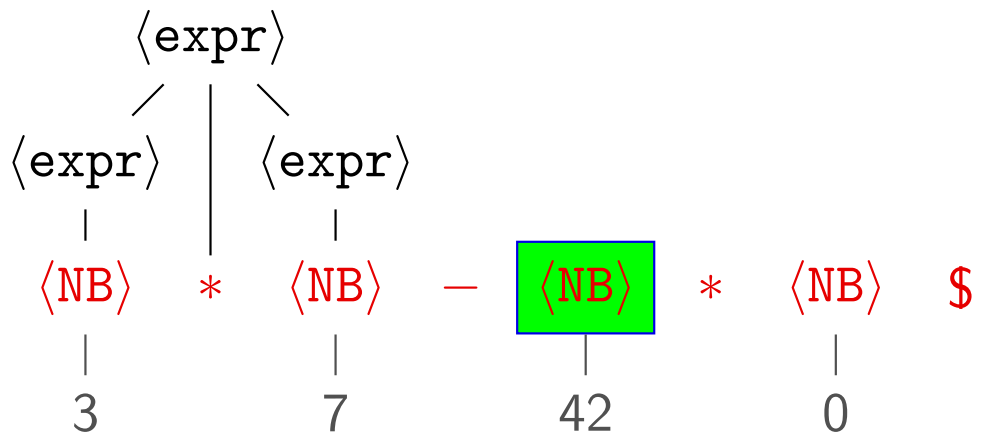
Second attempt in motion



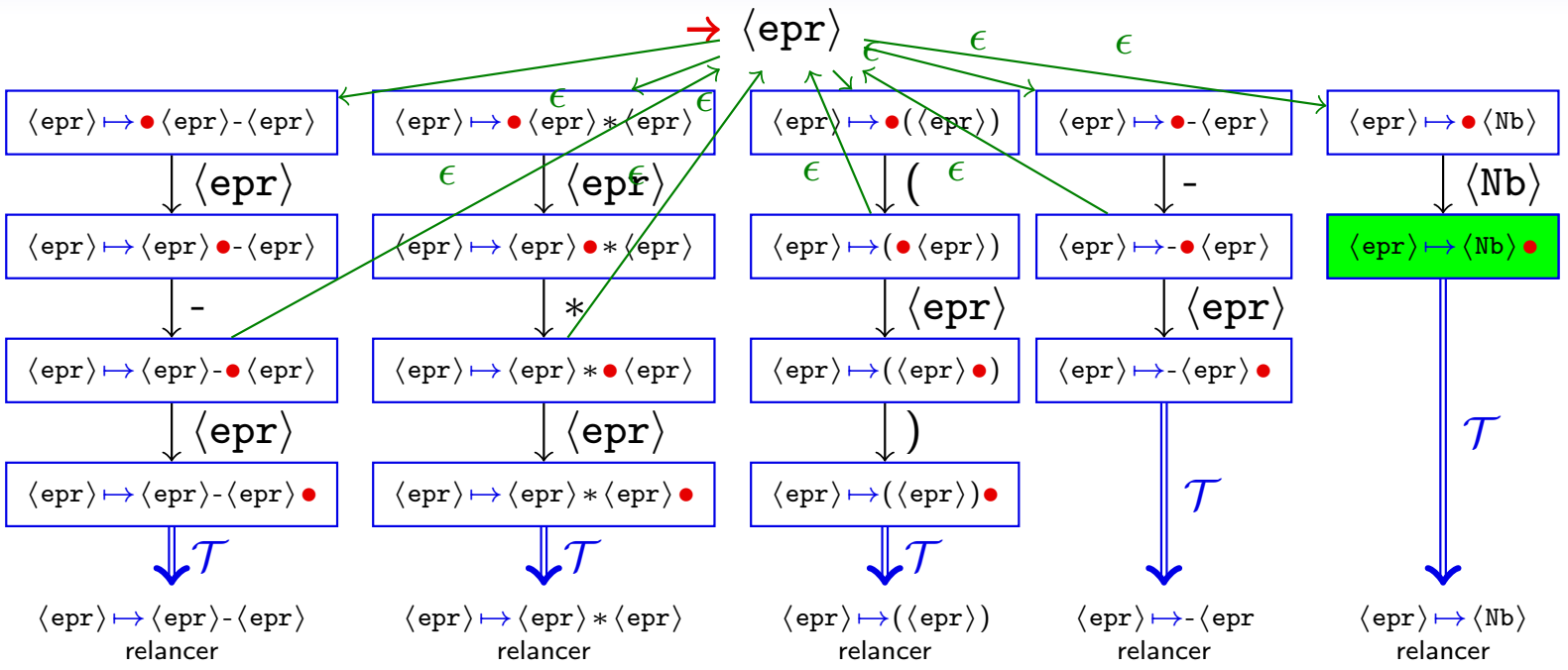


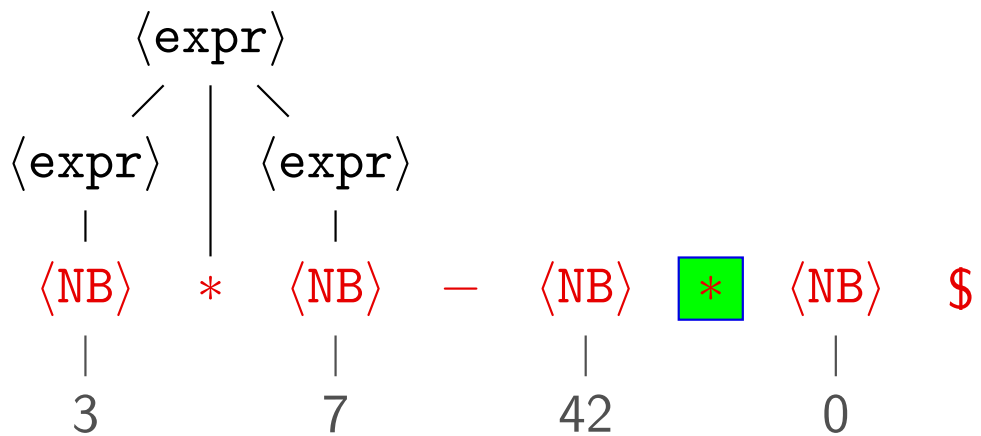
Second attempt in motion



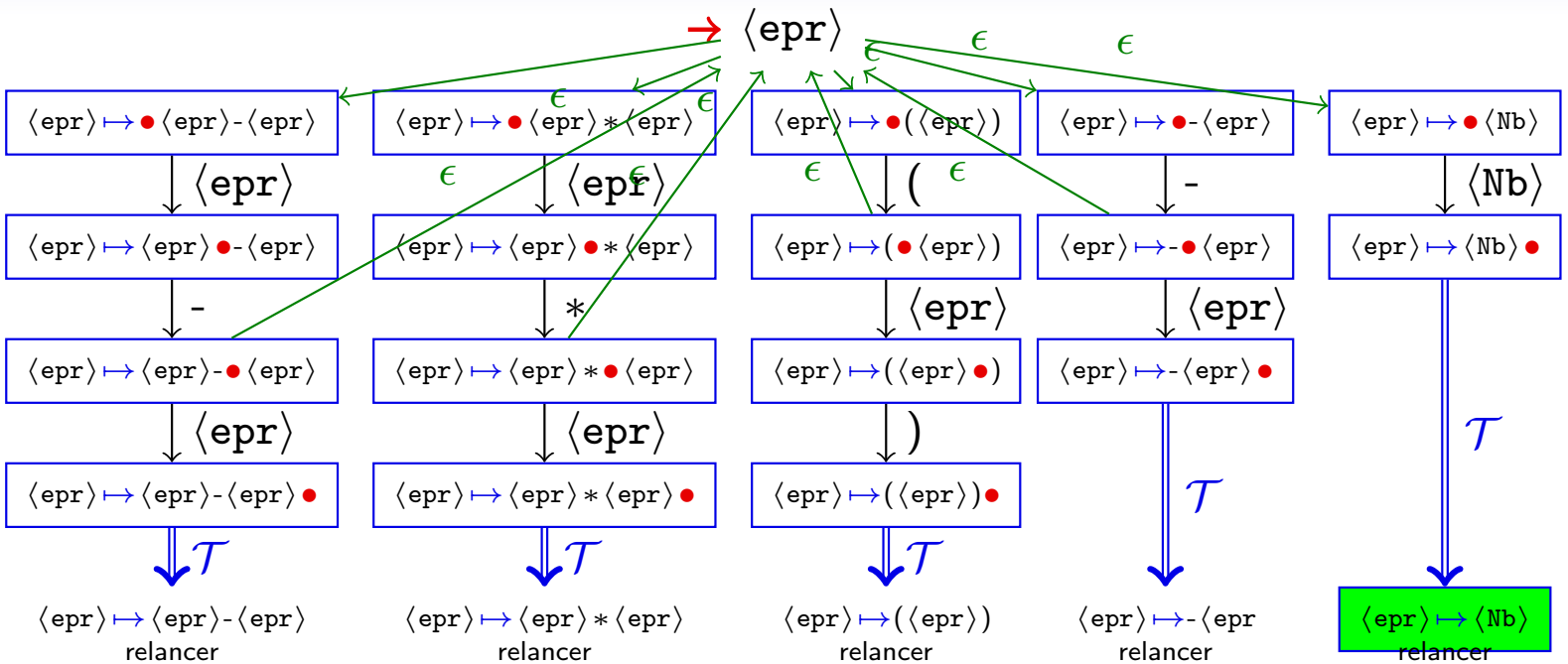


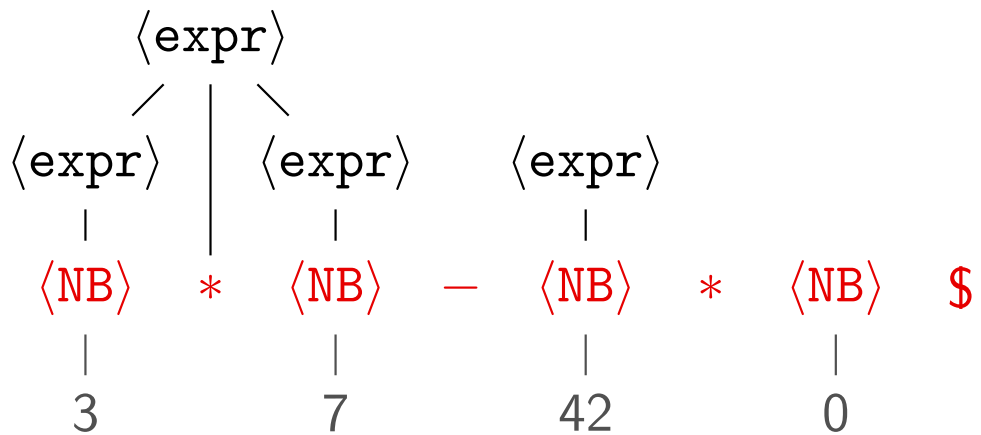
Second attempt in motion



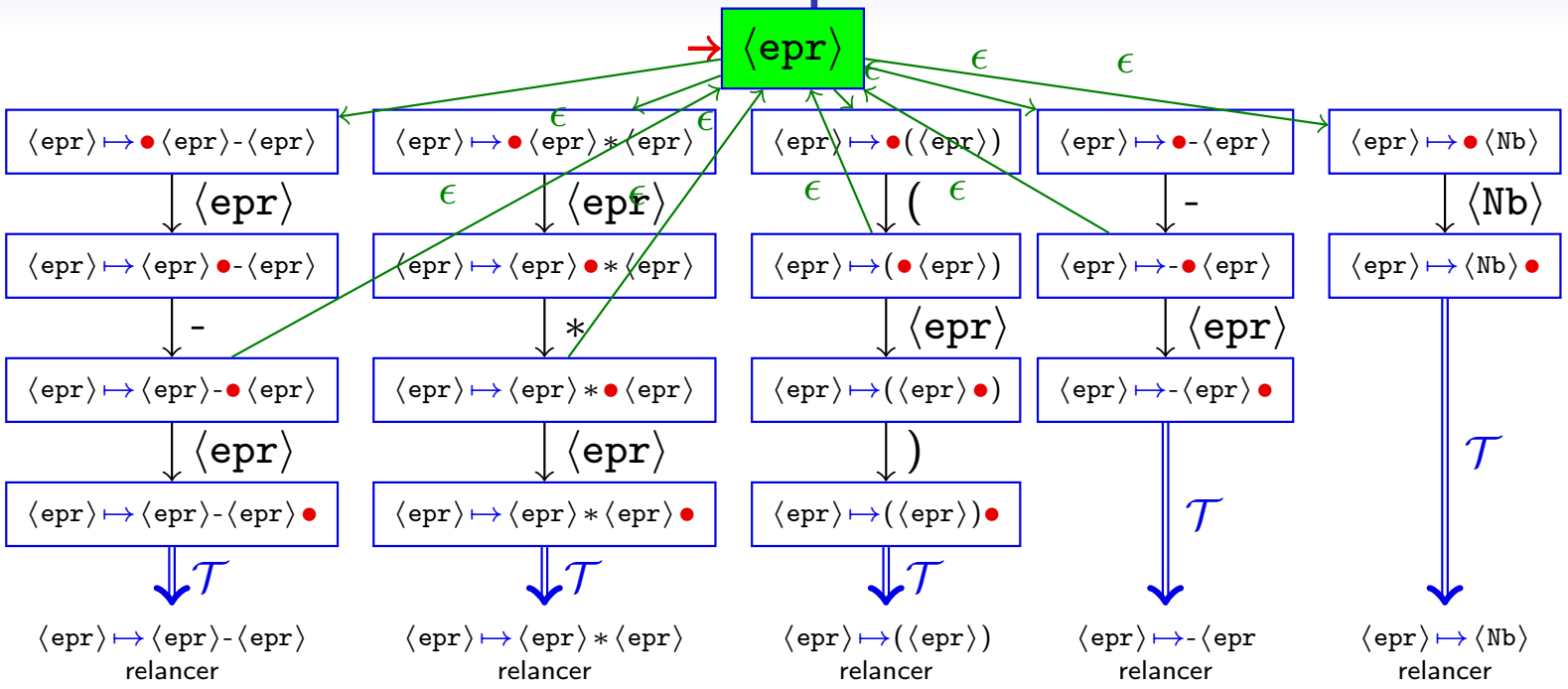


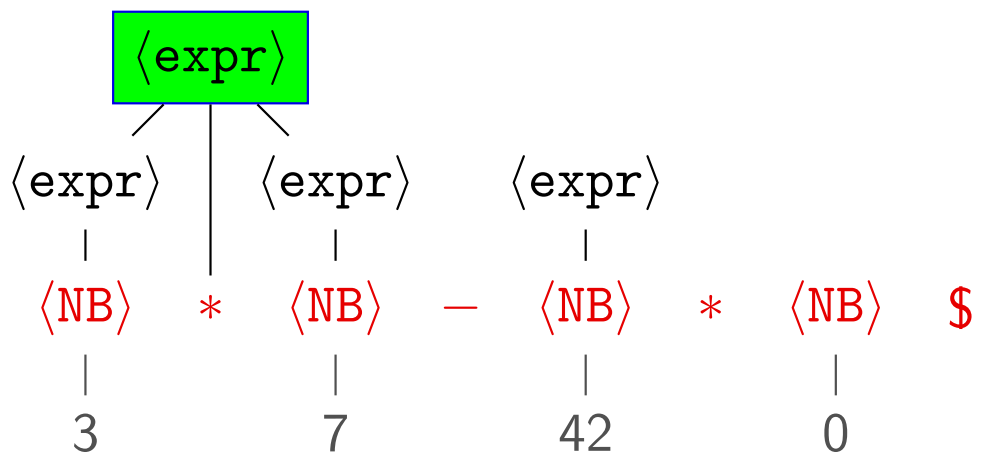
Second attempt in motion



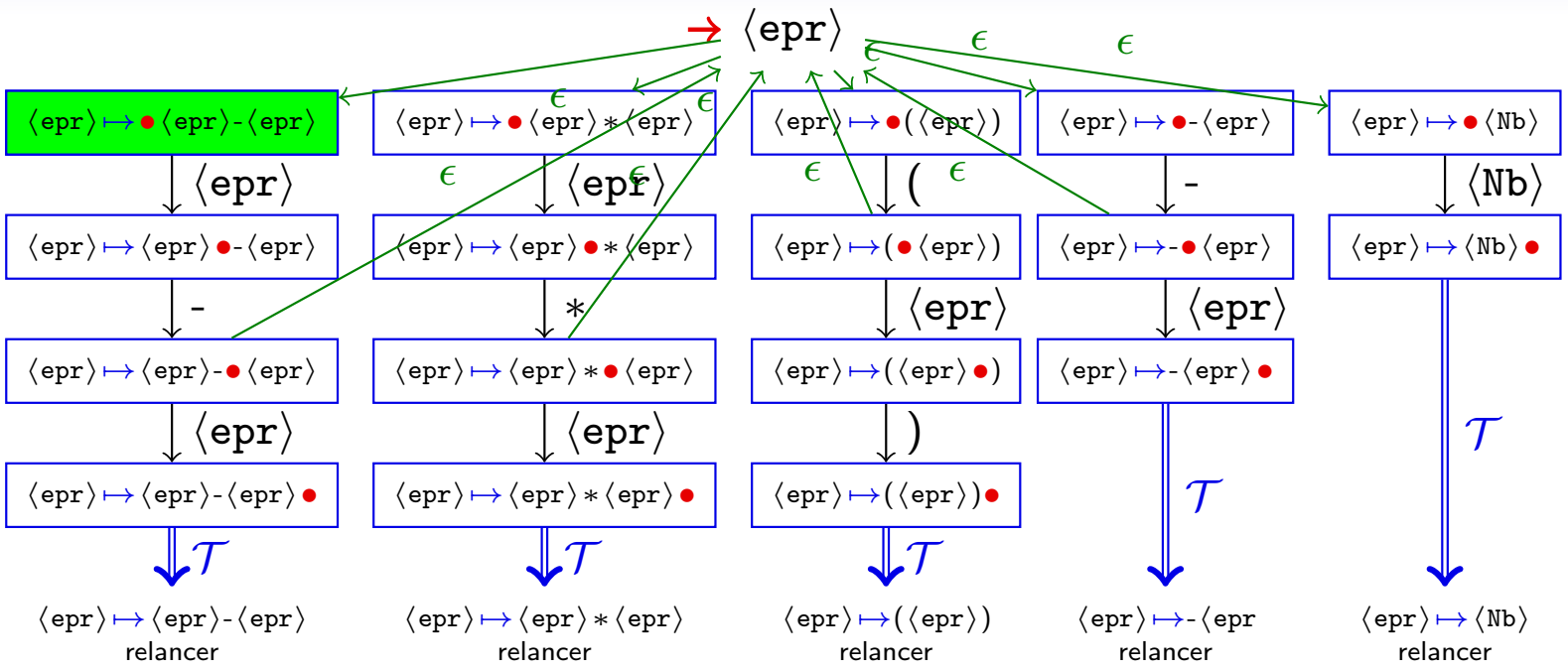


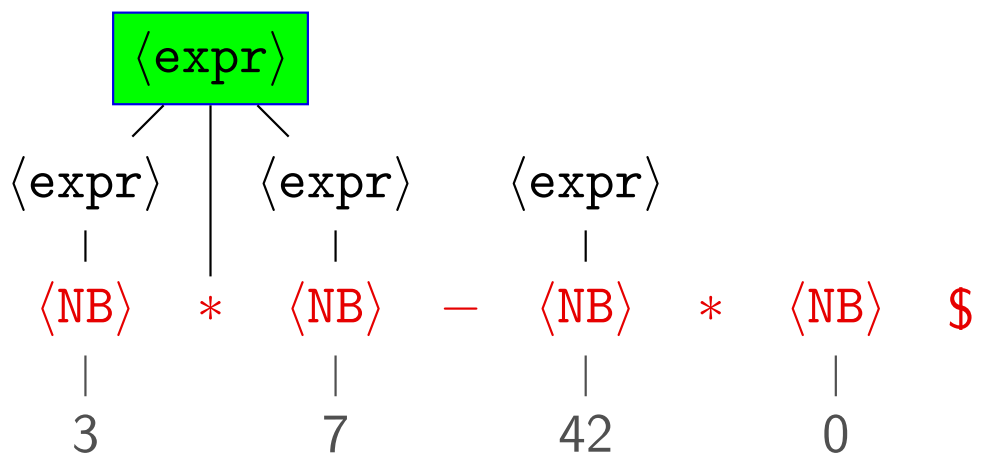
Second attempt in motion



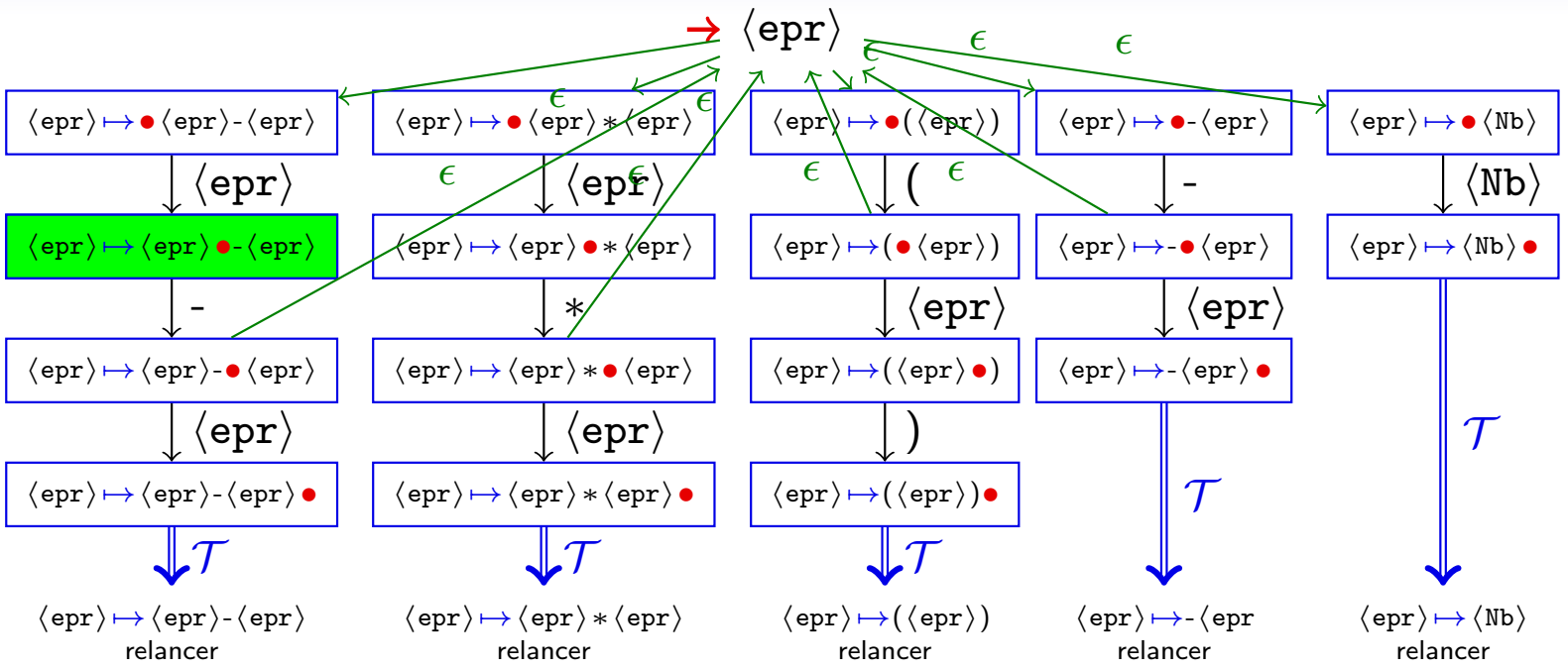


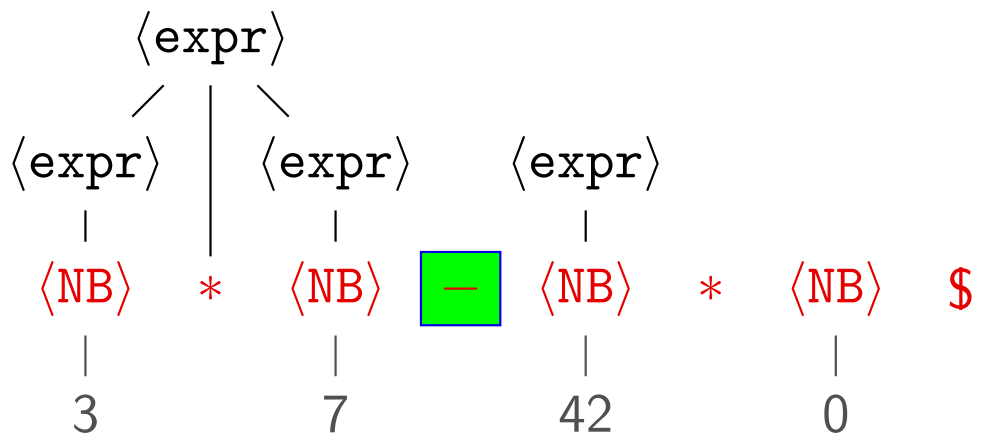
Second attempt in motion



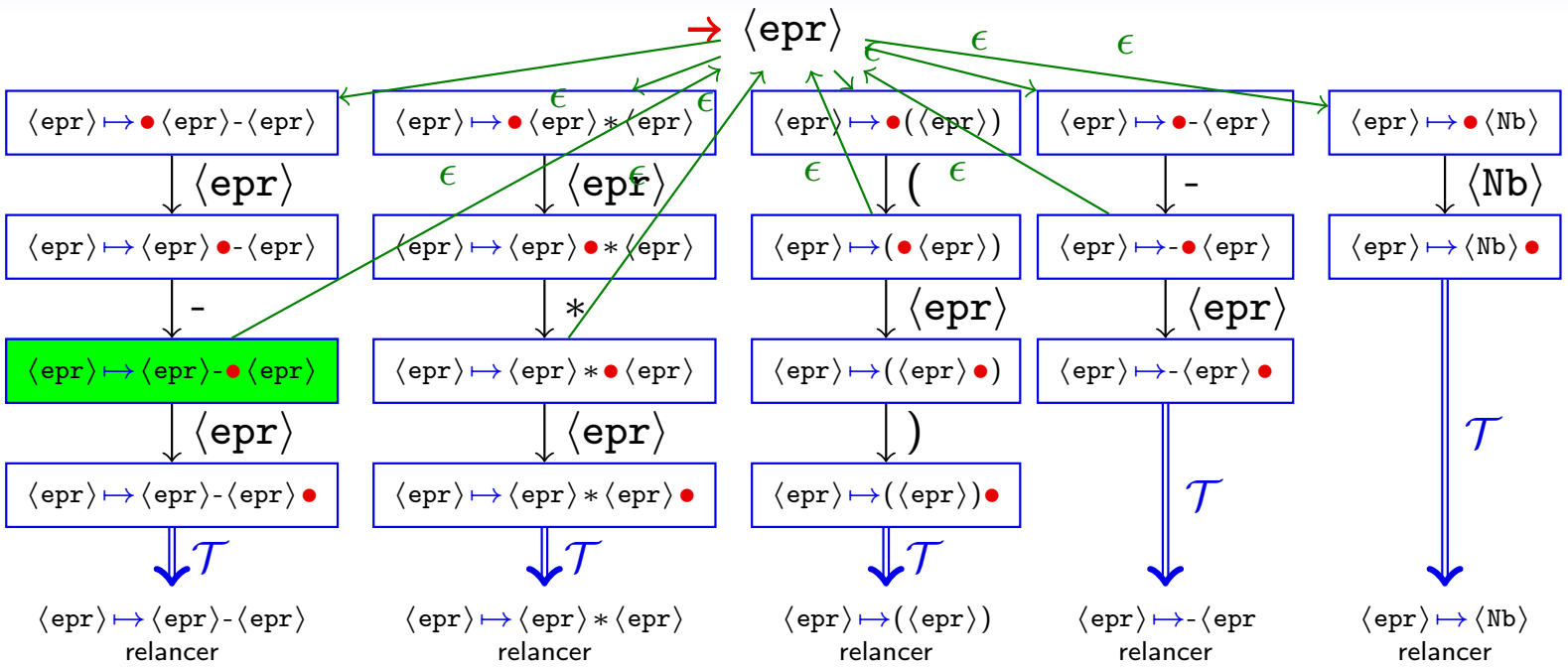


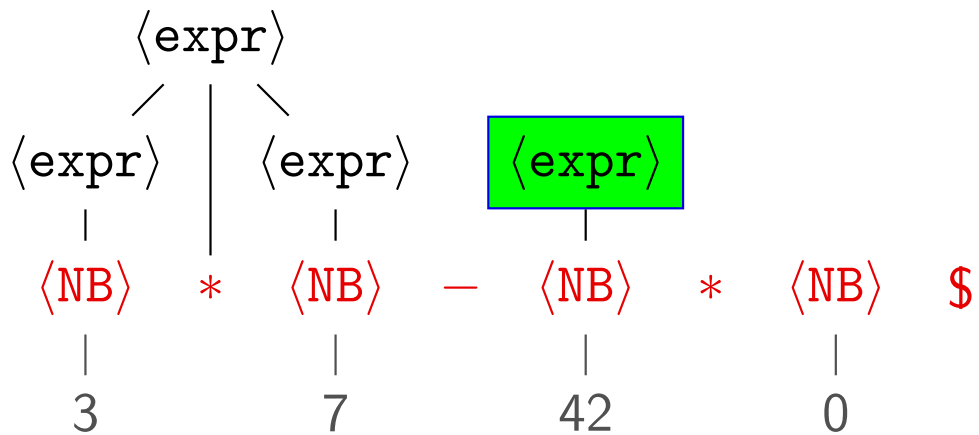
Second attempt in motion



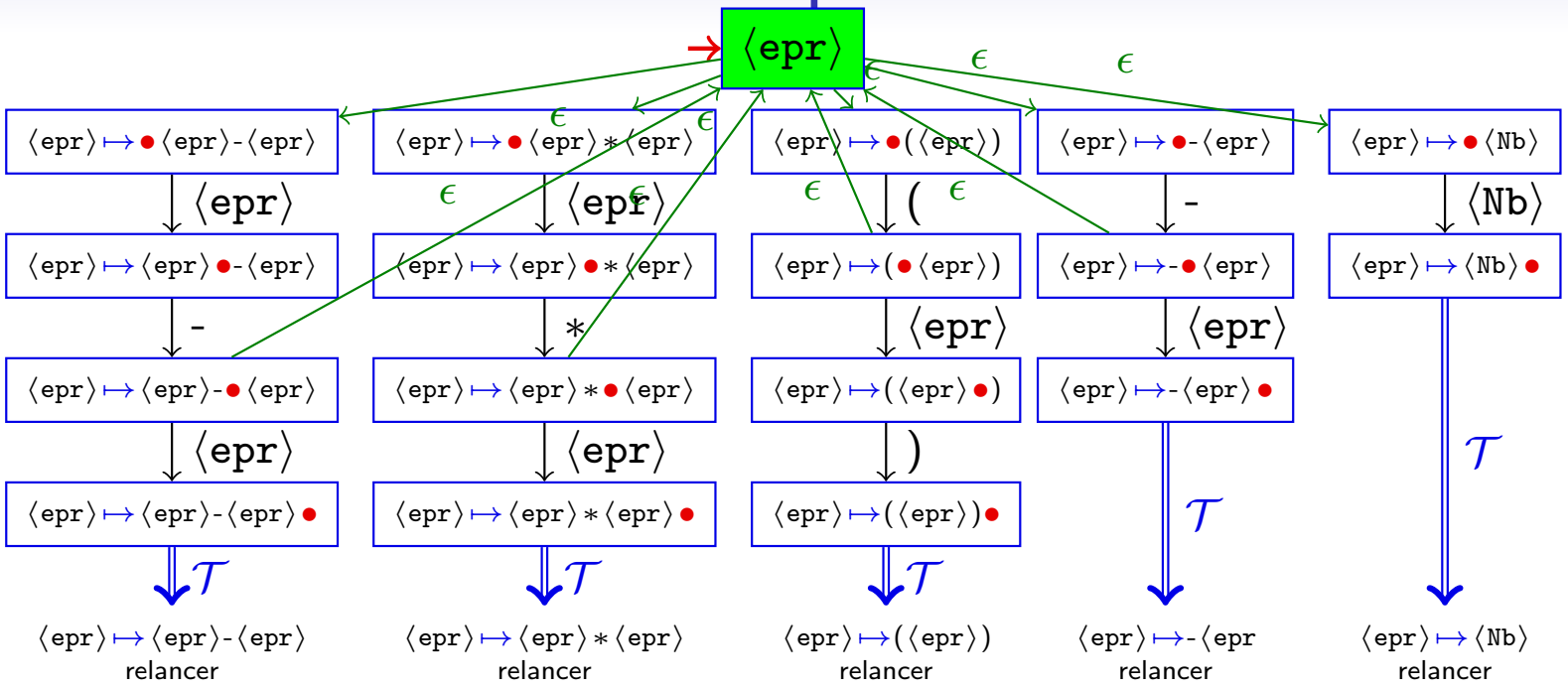


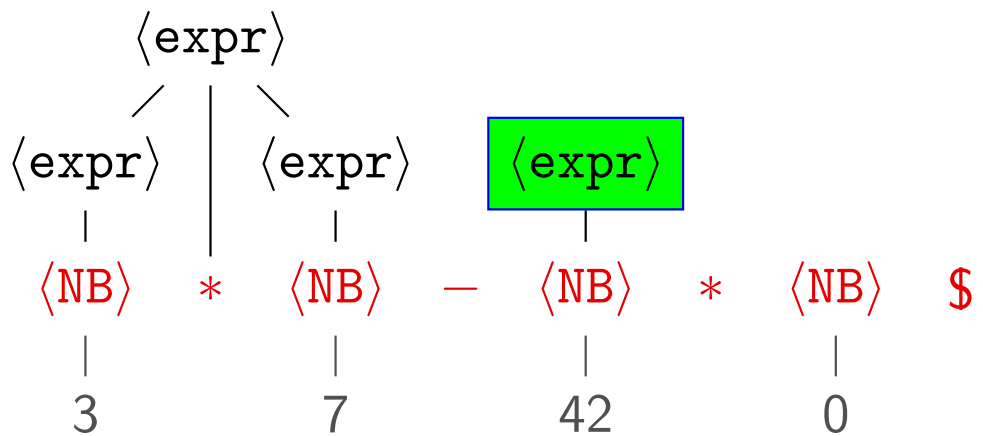
Second attempt in motion



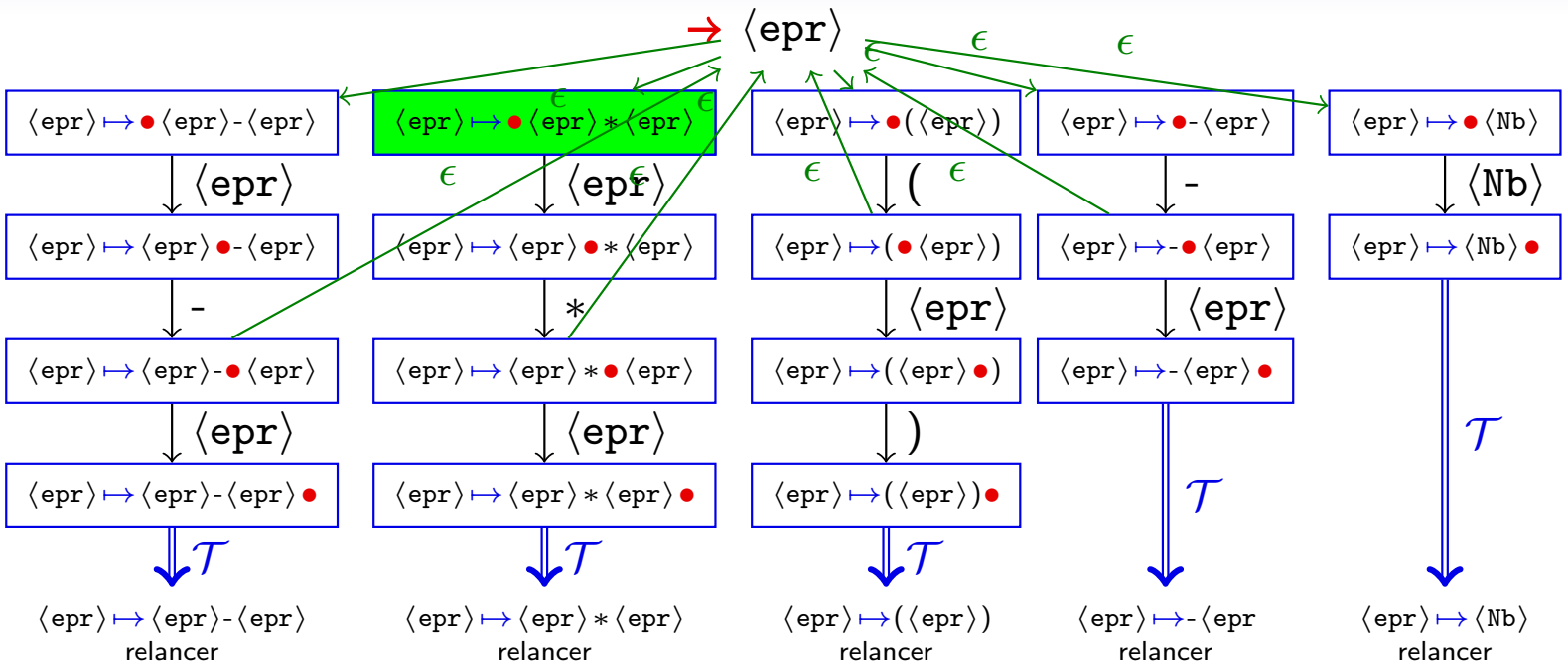


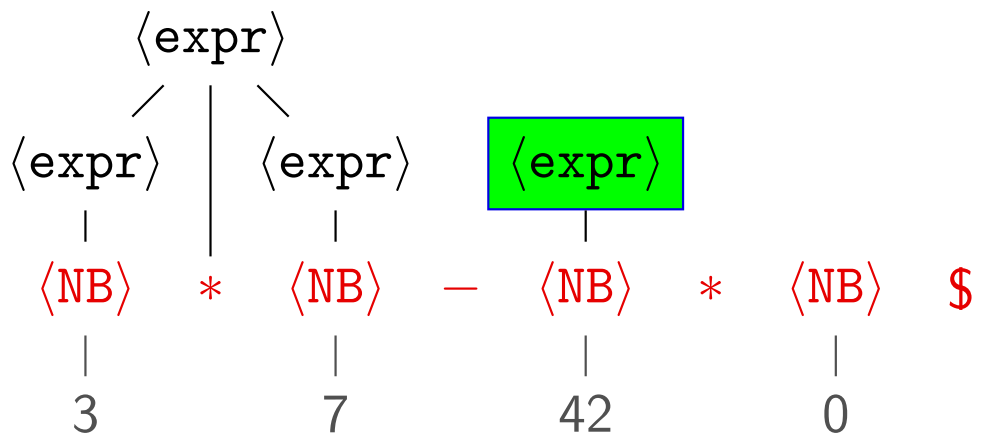
Second attempt in motion



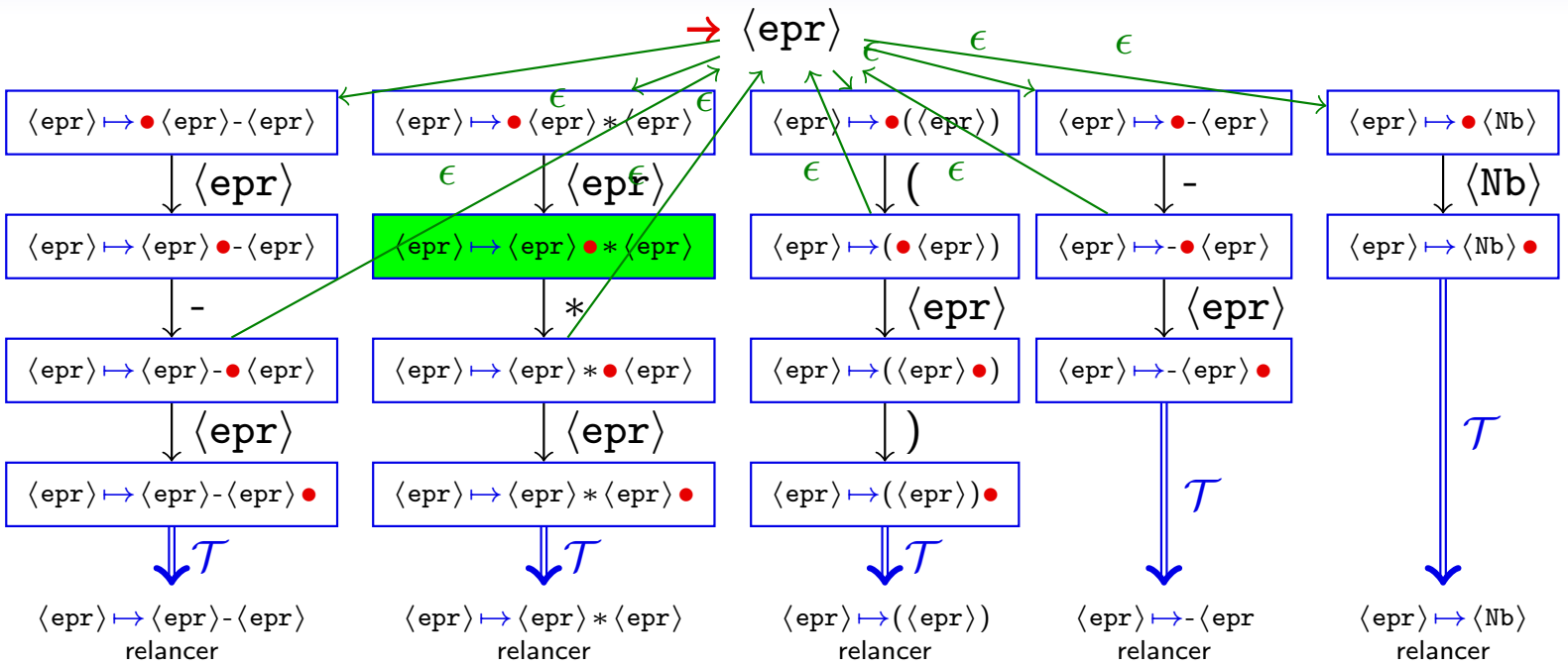


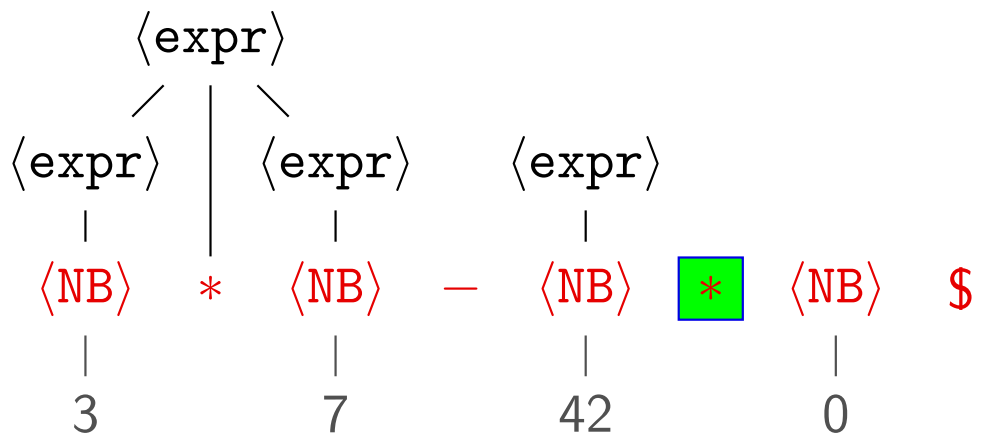
Second attempt in motion



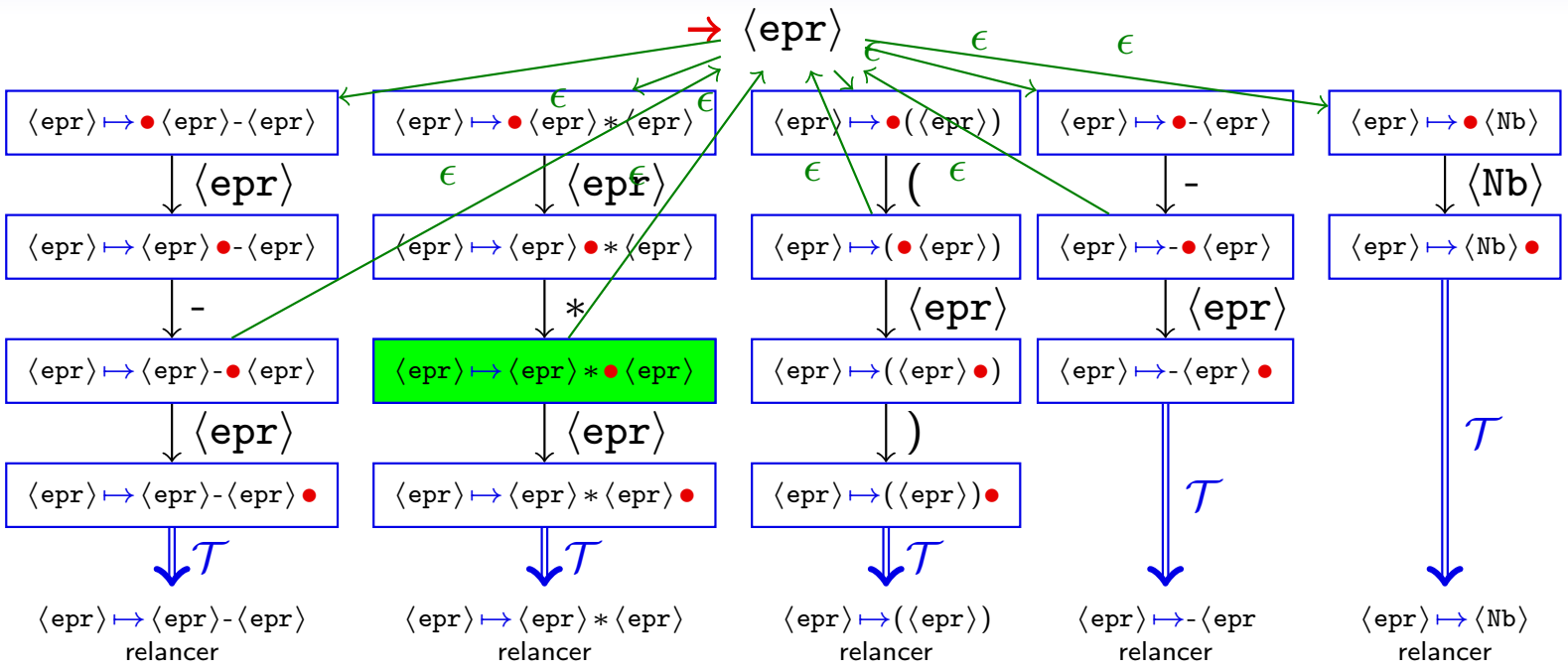


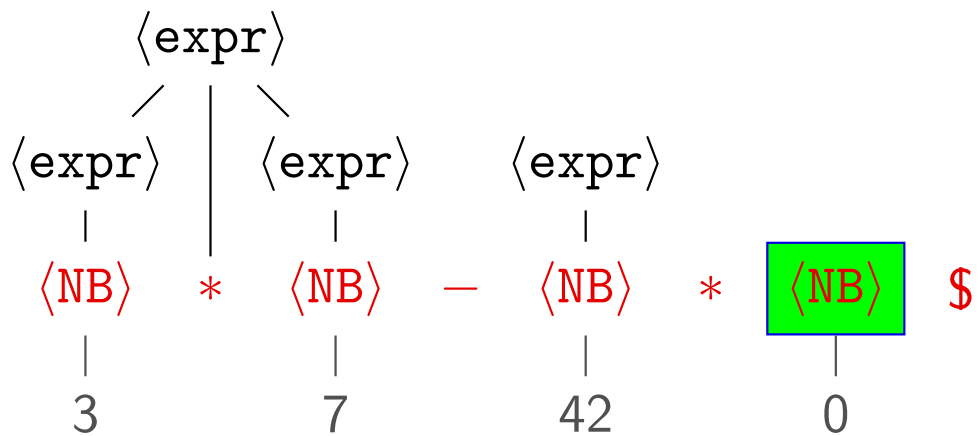
Second attempt in motion



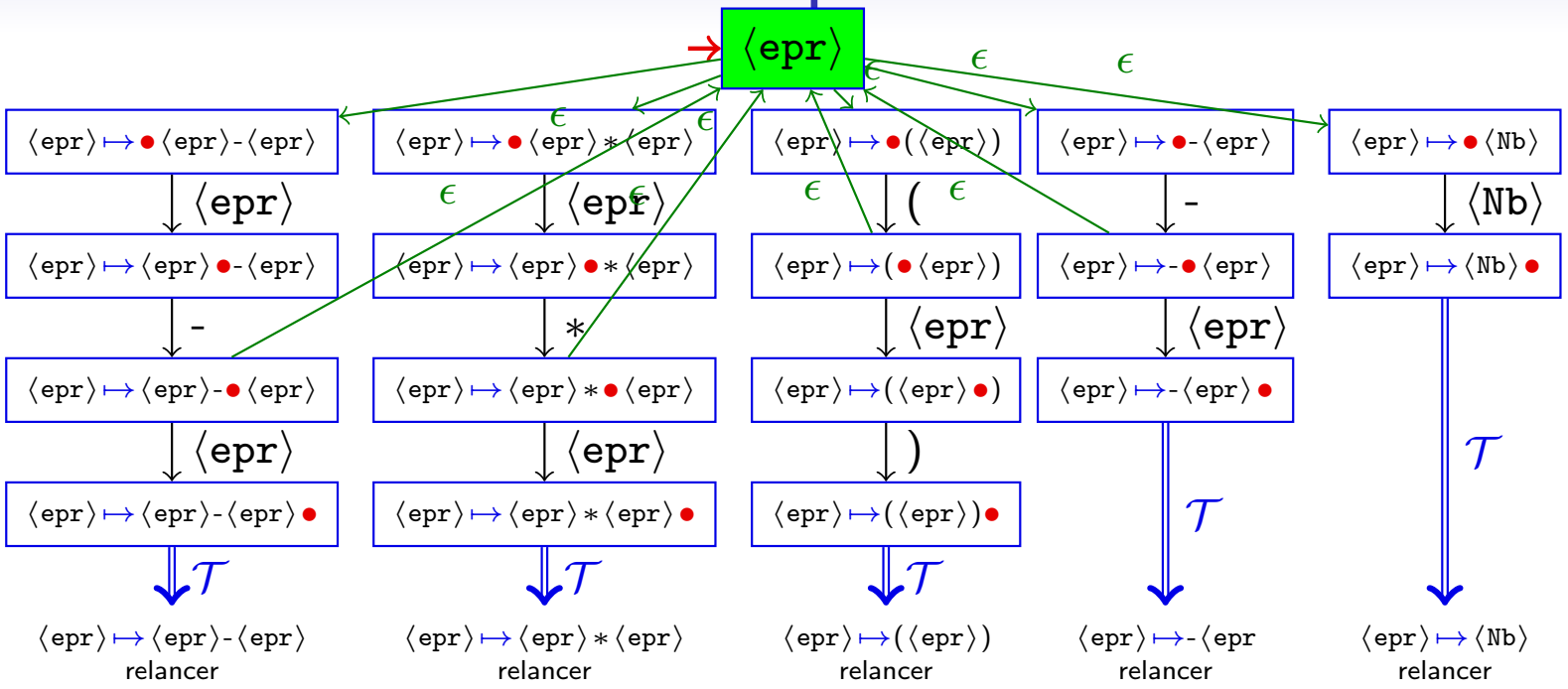


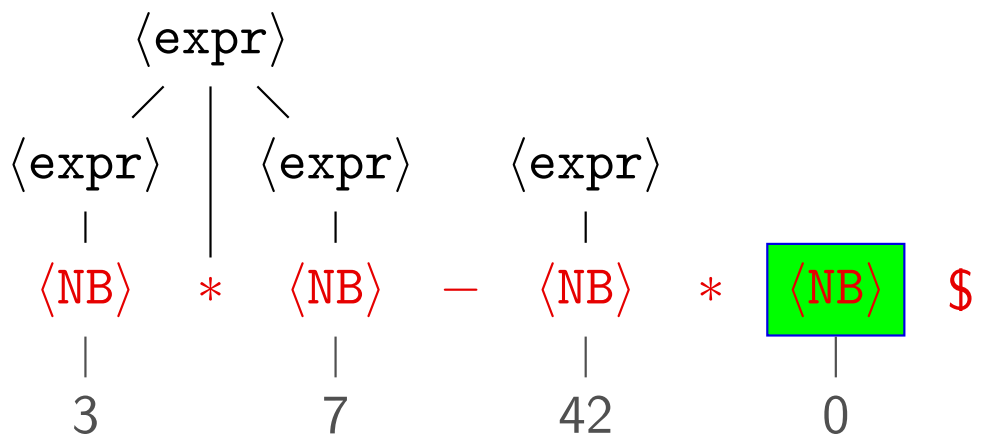
Second attempt in motion



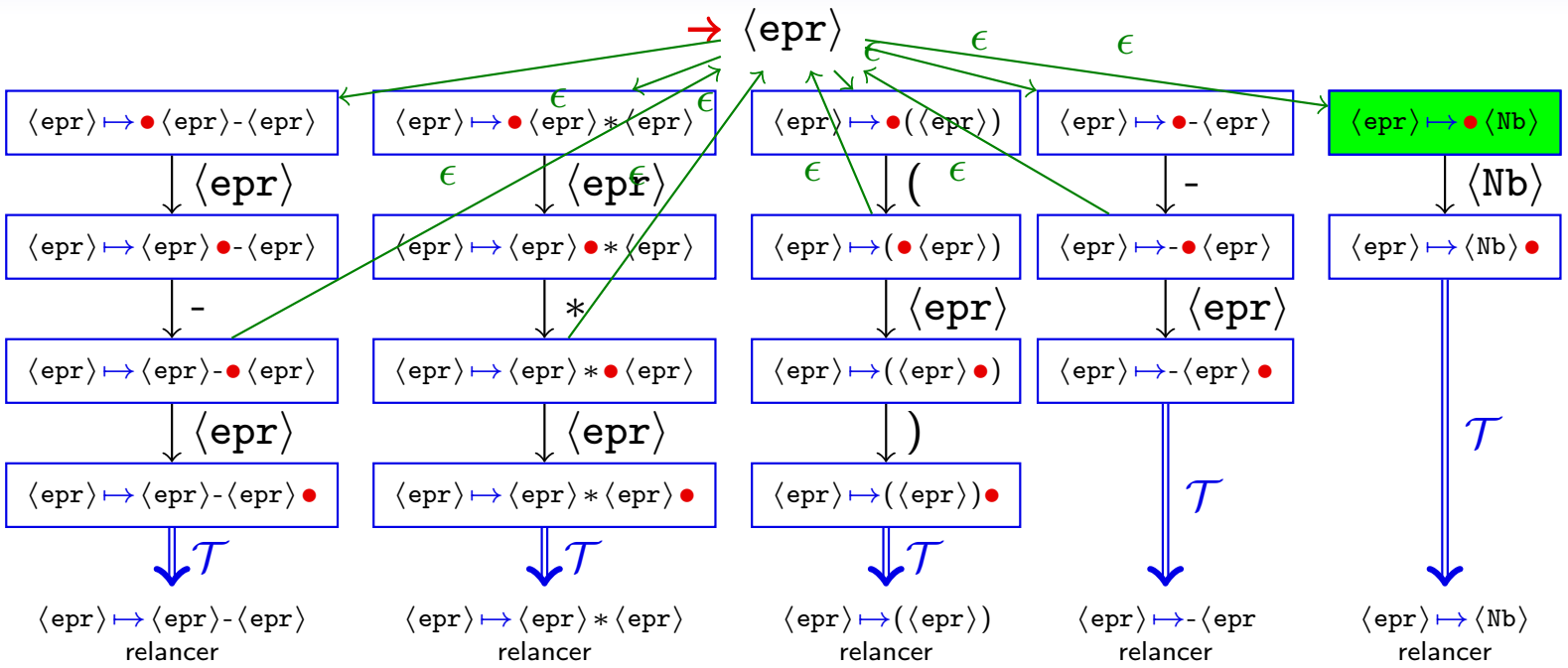


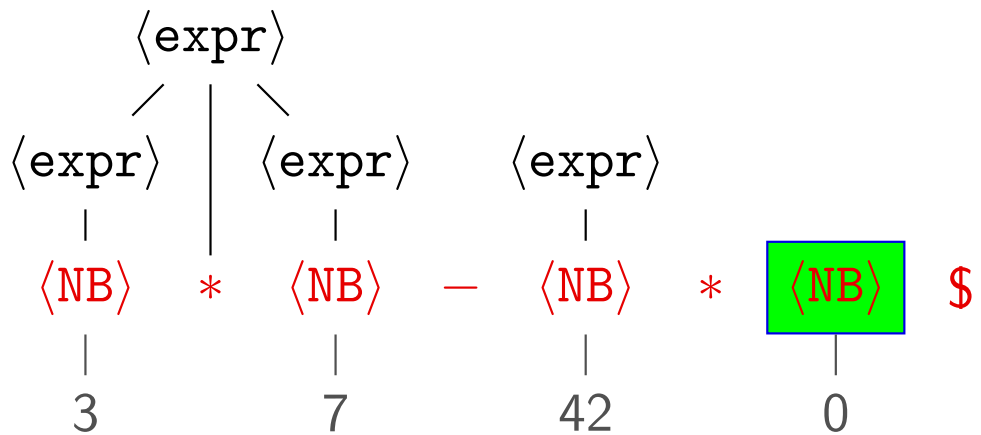
Second attempt in motion



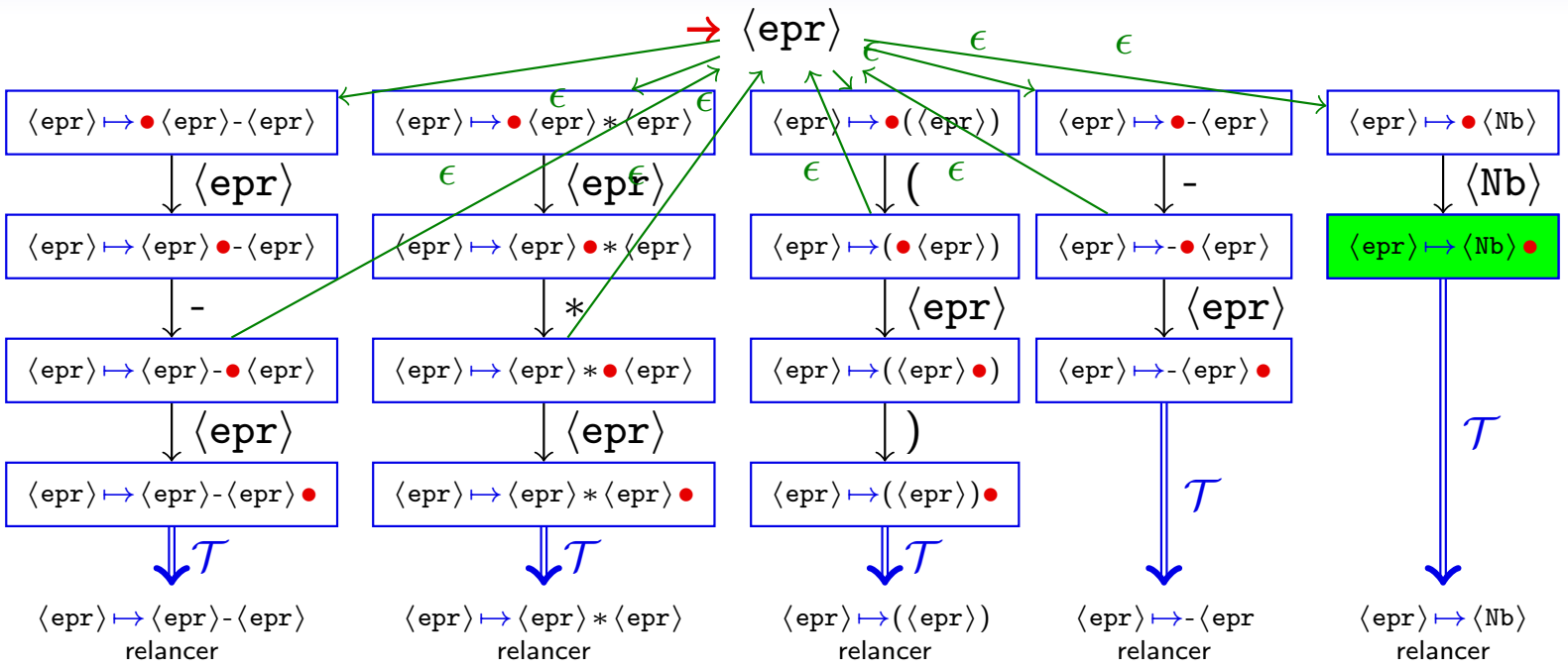


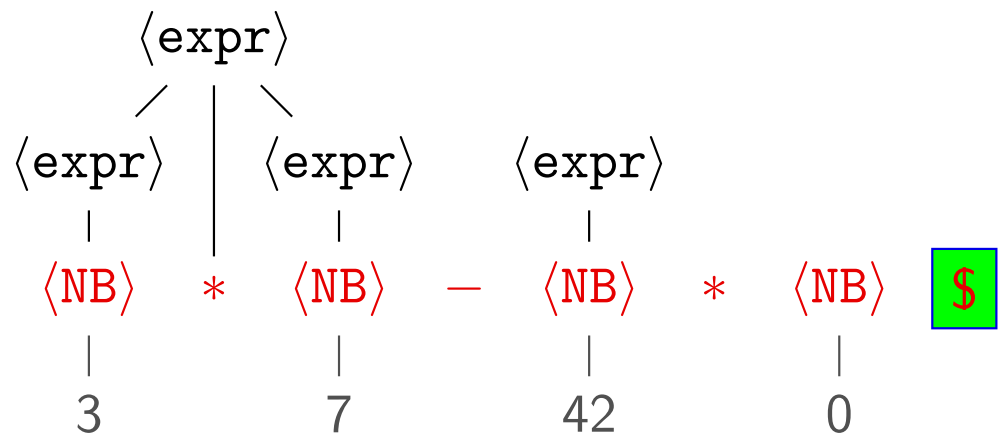
Second attempt in motion



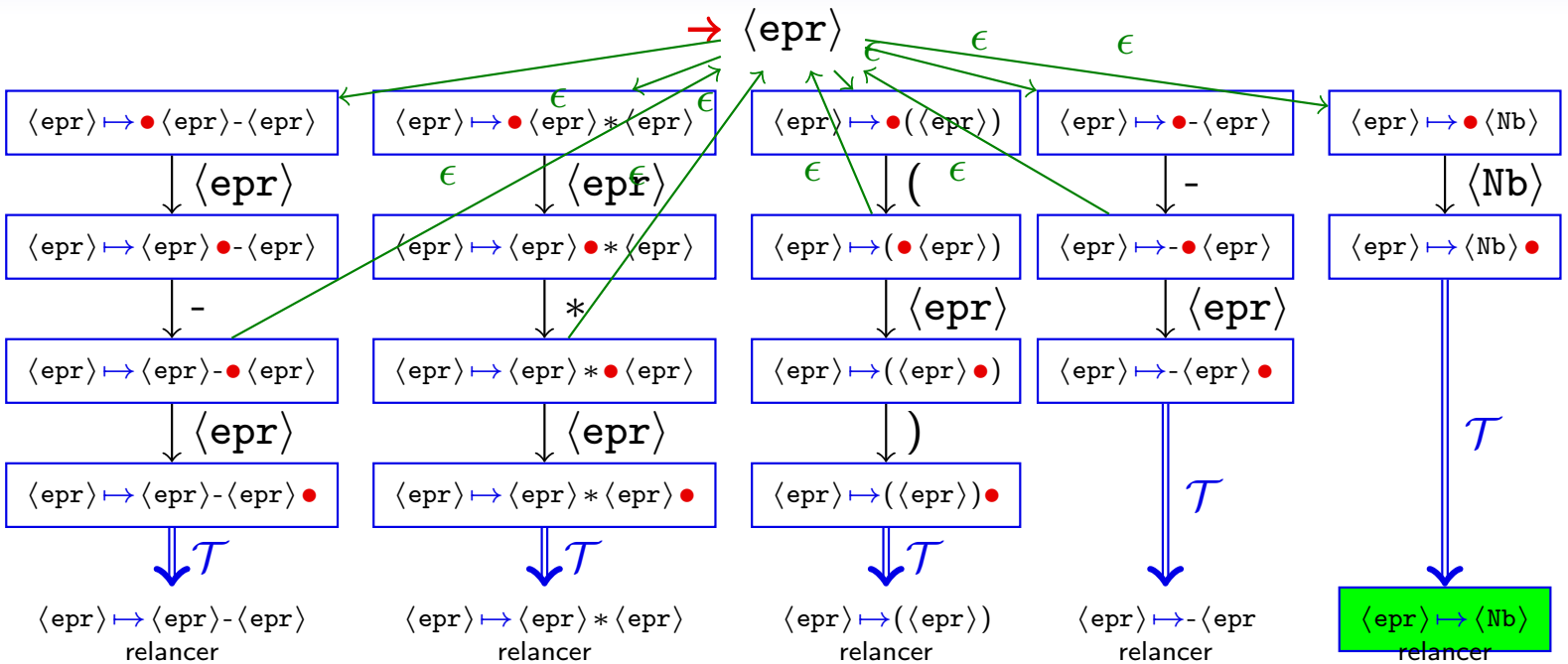


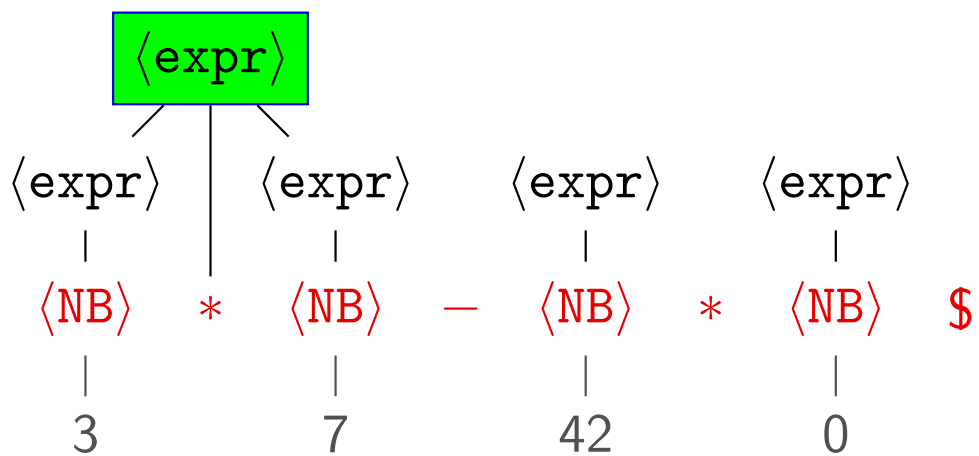
Second attempt in motion



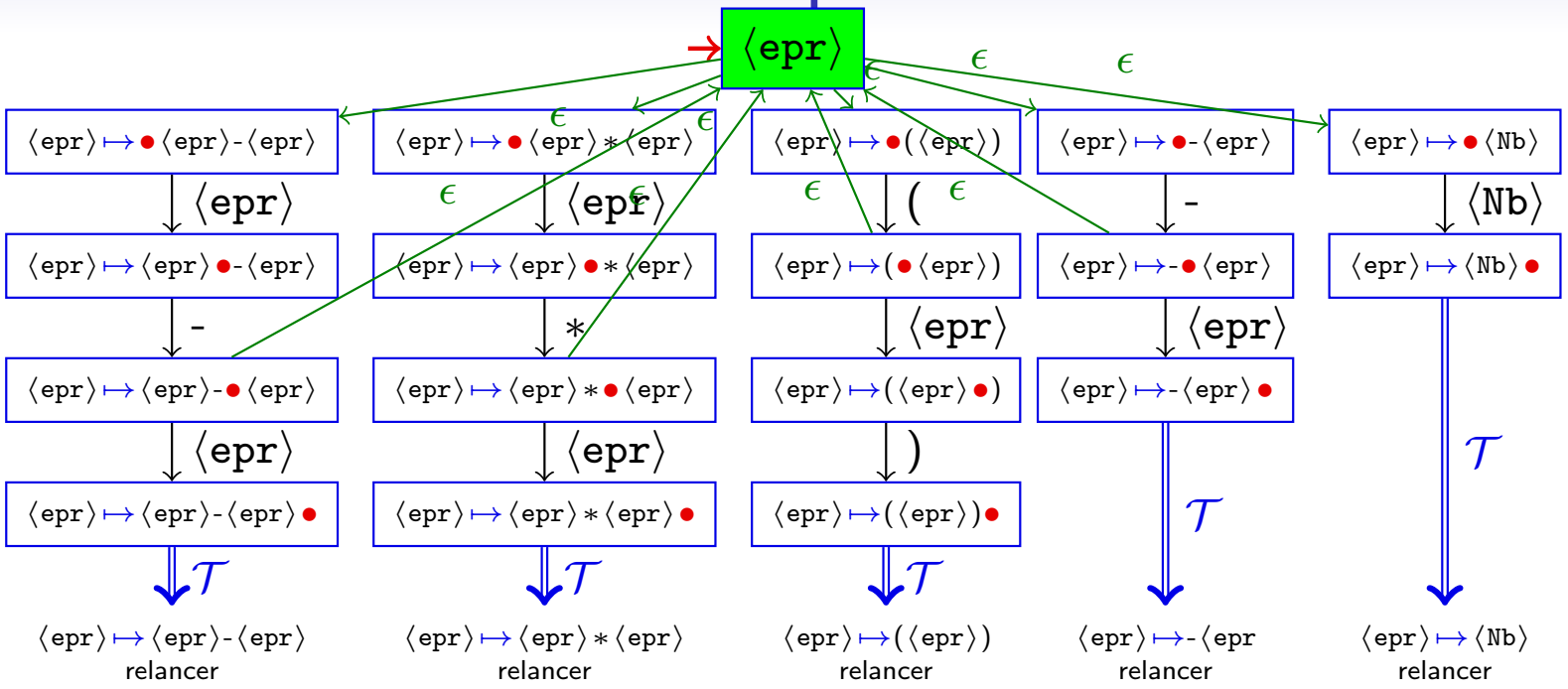


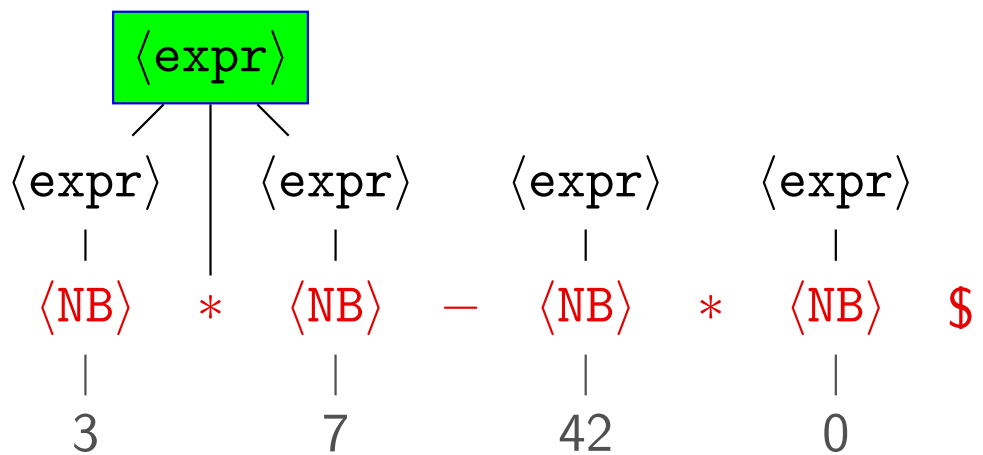
Second attempt in motion



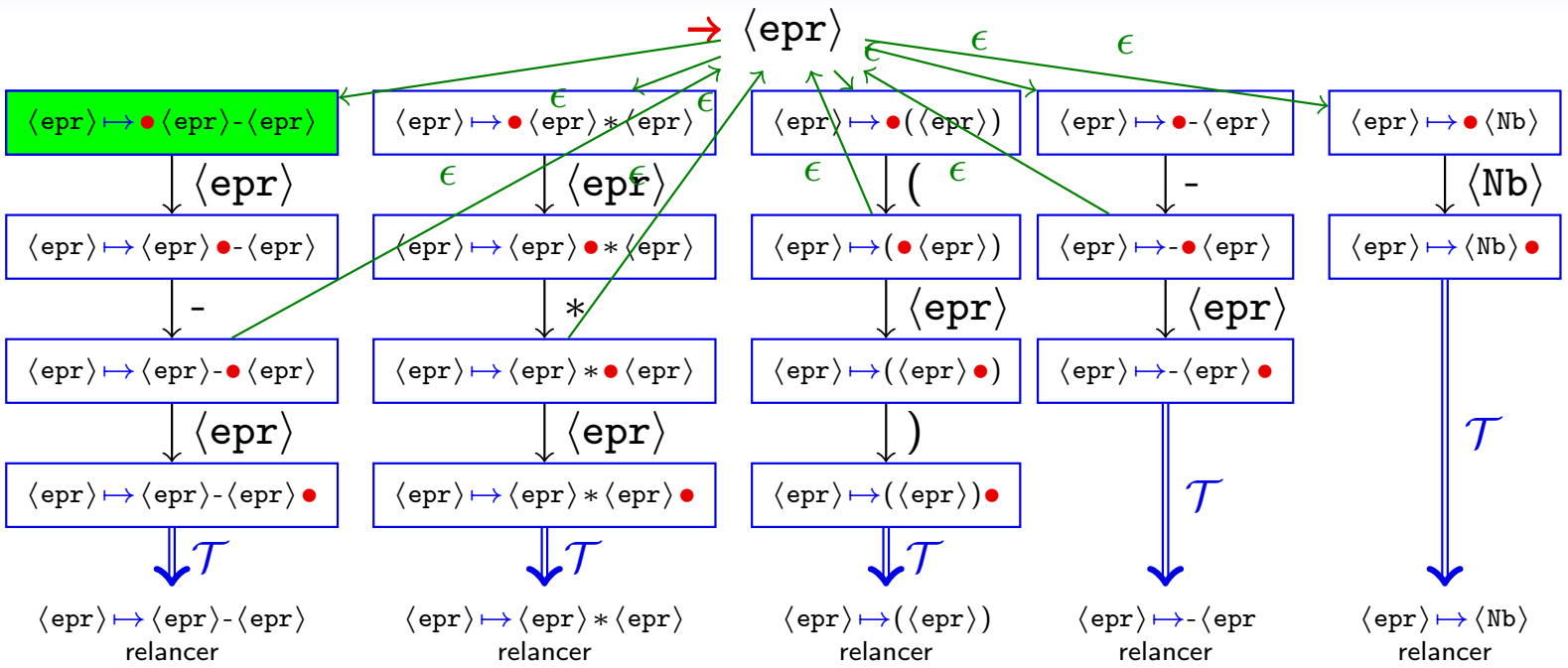


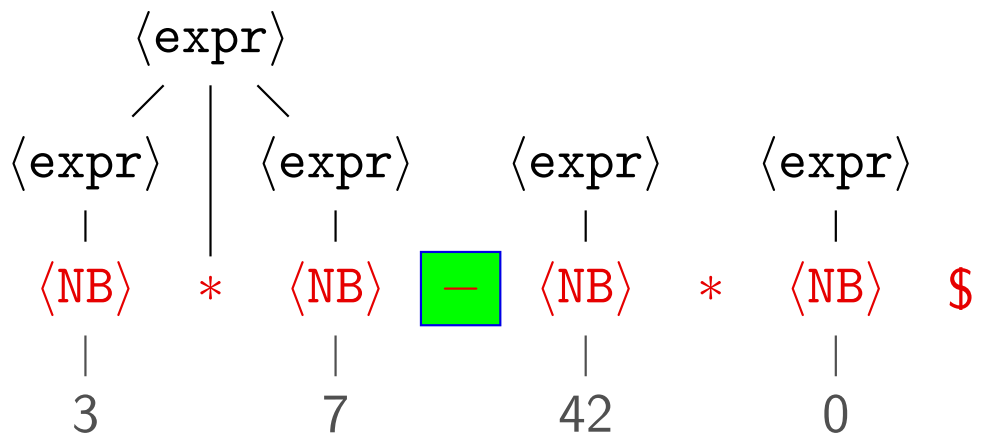
Second attempt in motion



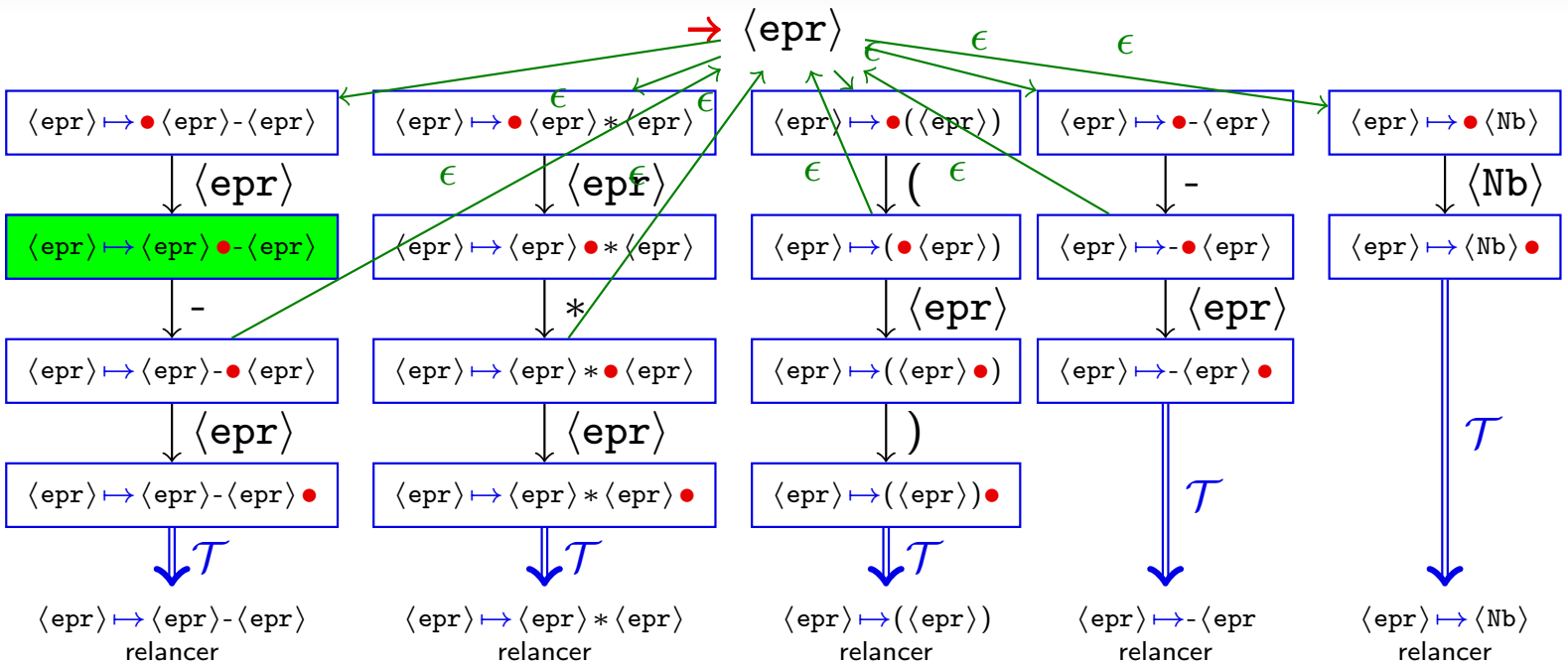


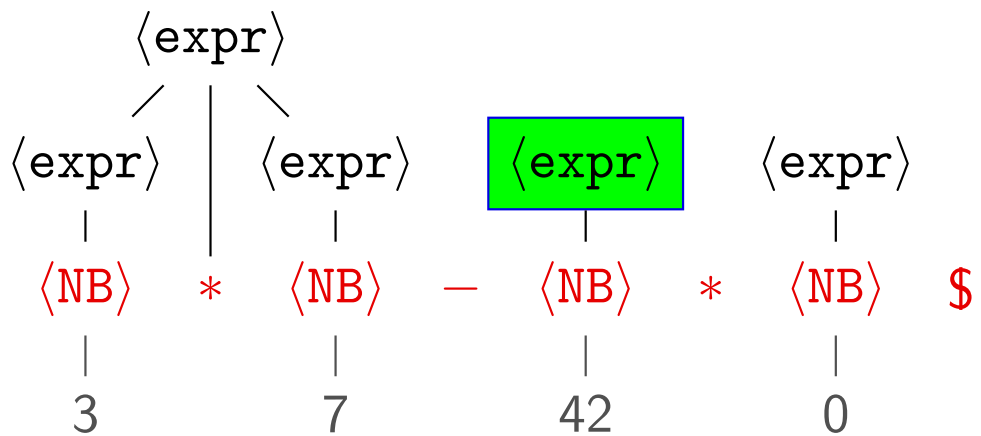
Second attempt in motion



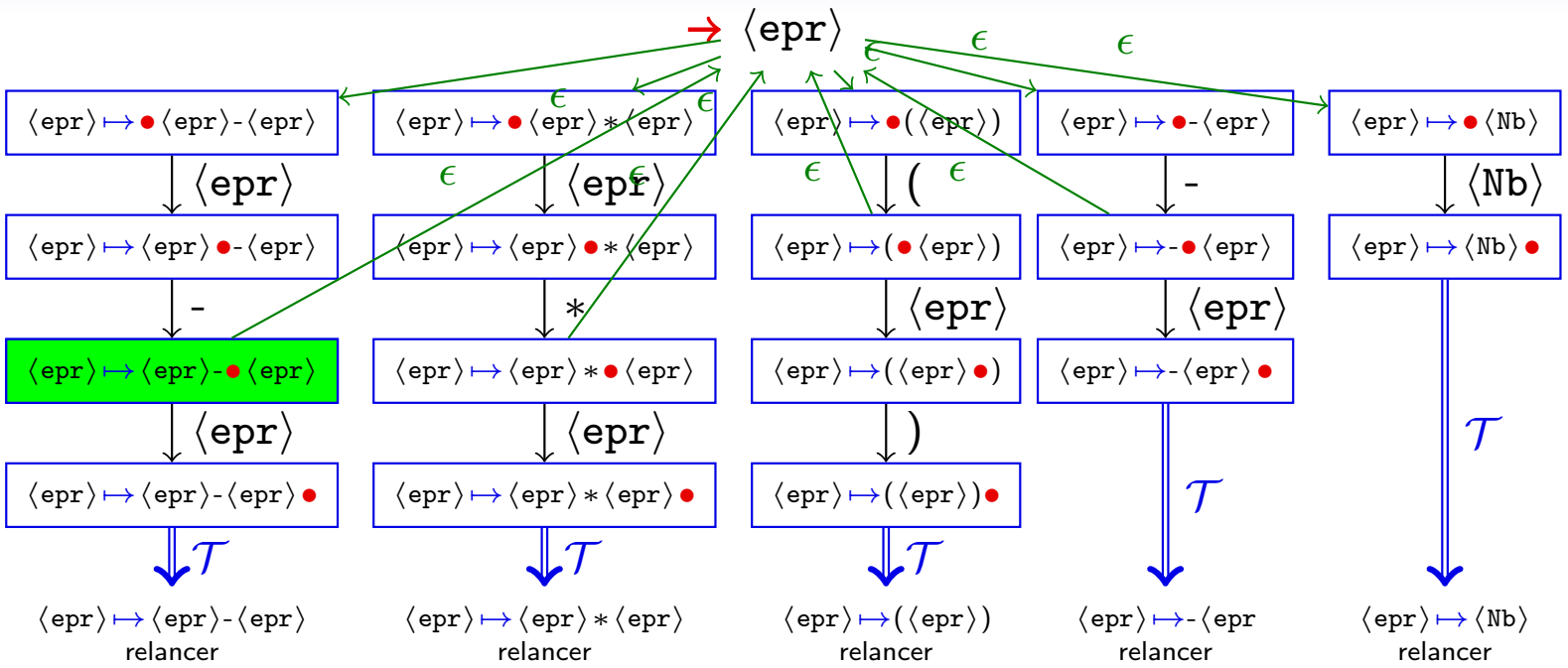


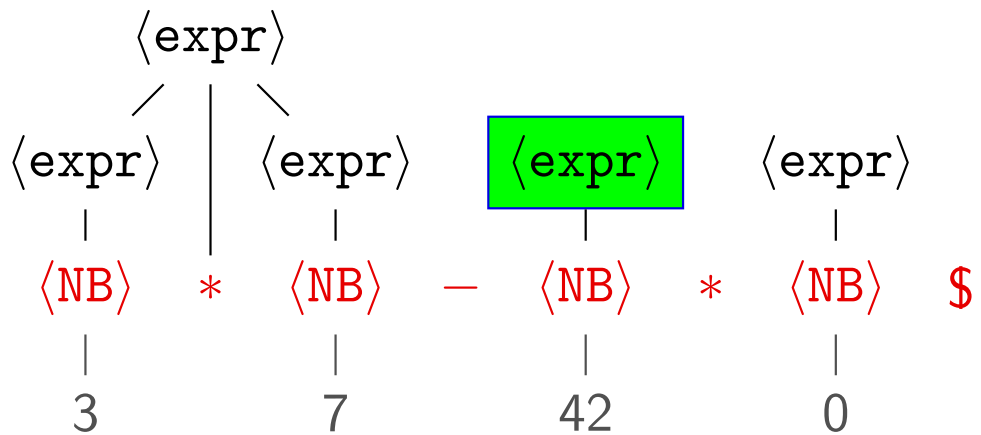
Second attempt in motion



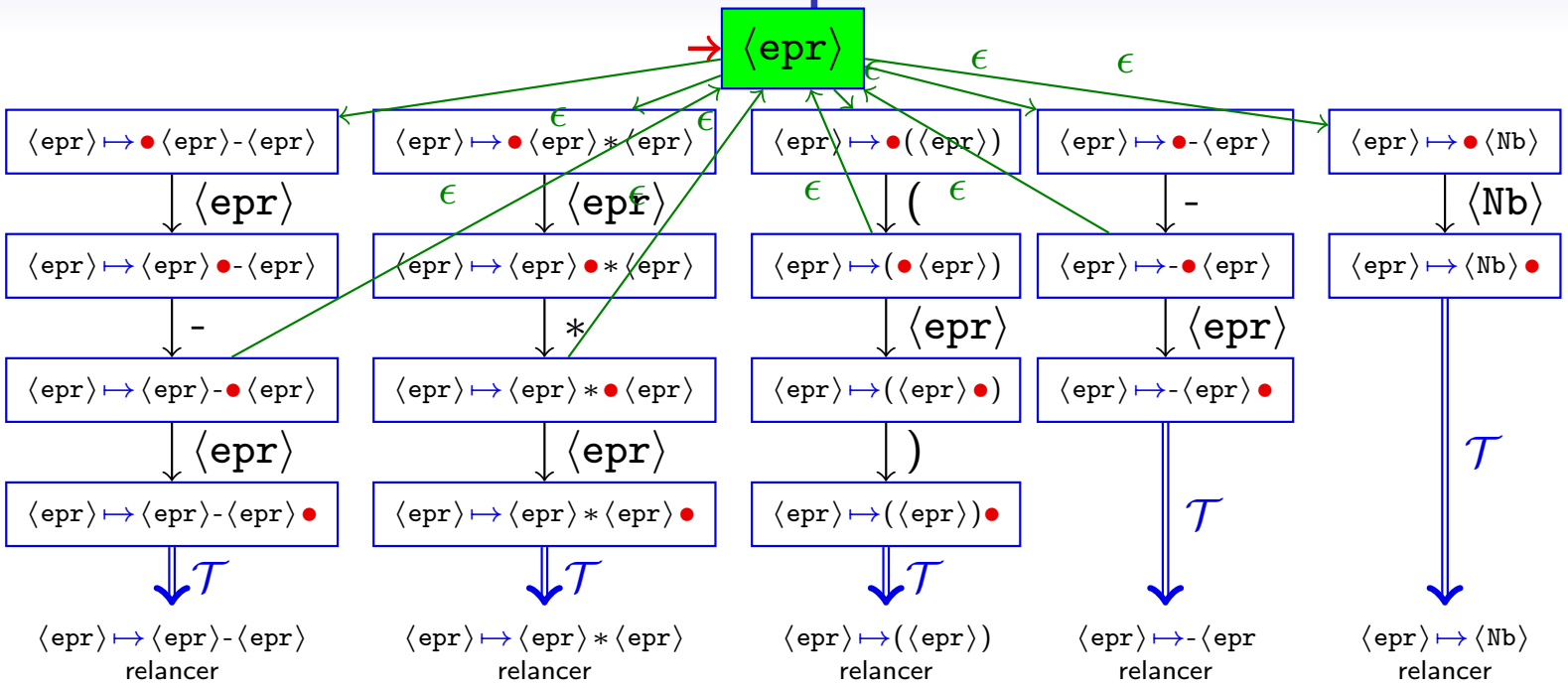


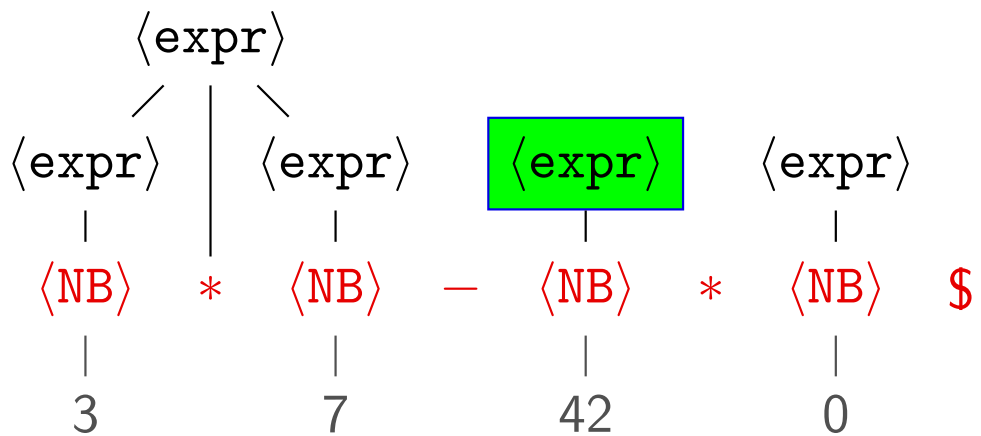
Second attempt in motion



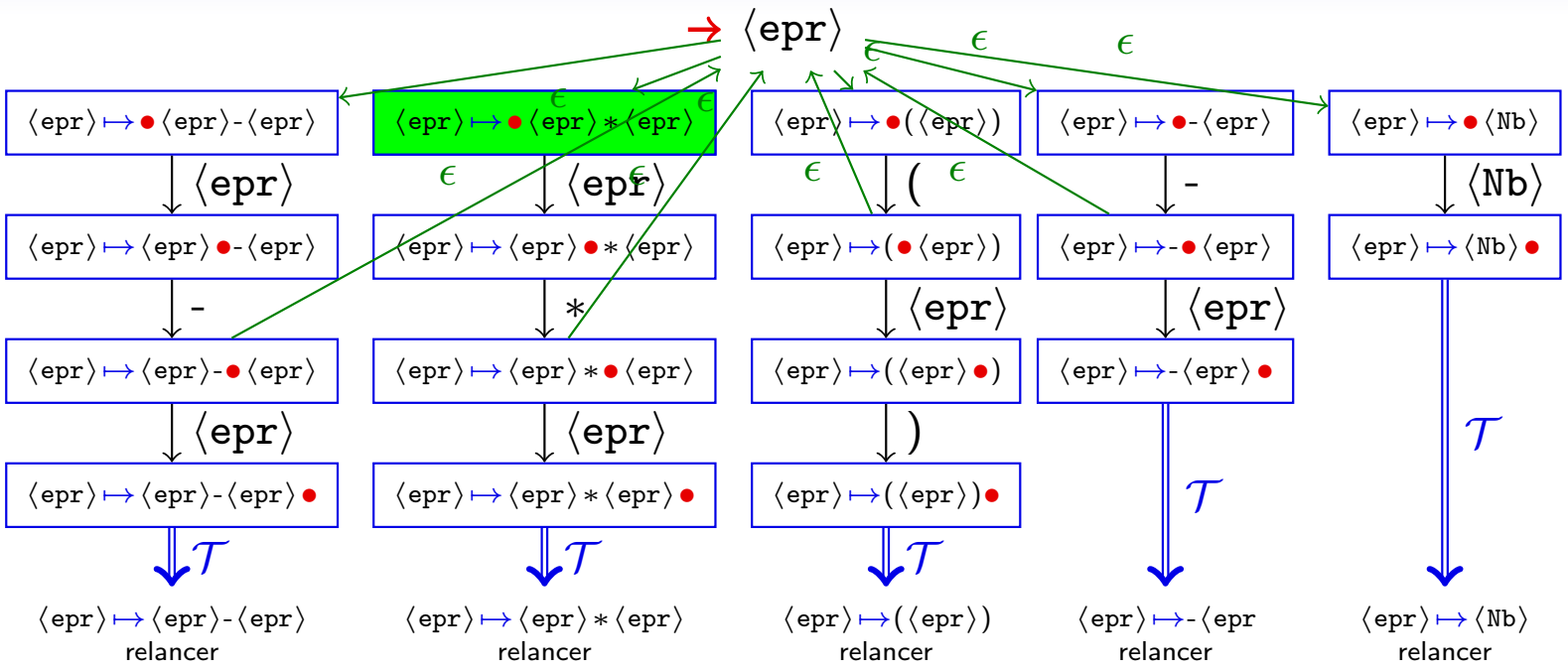


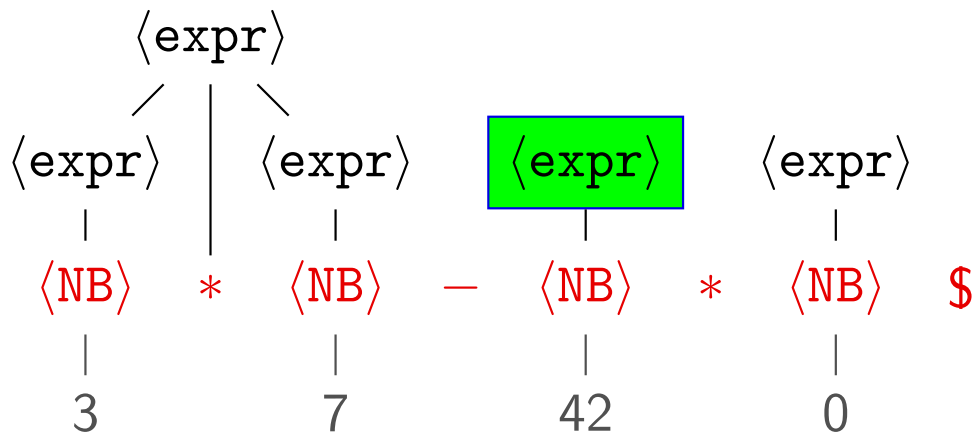
Second attempt in motion



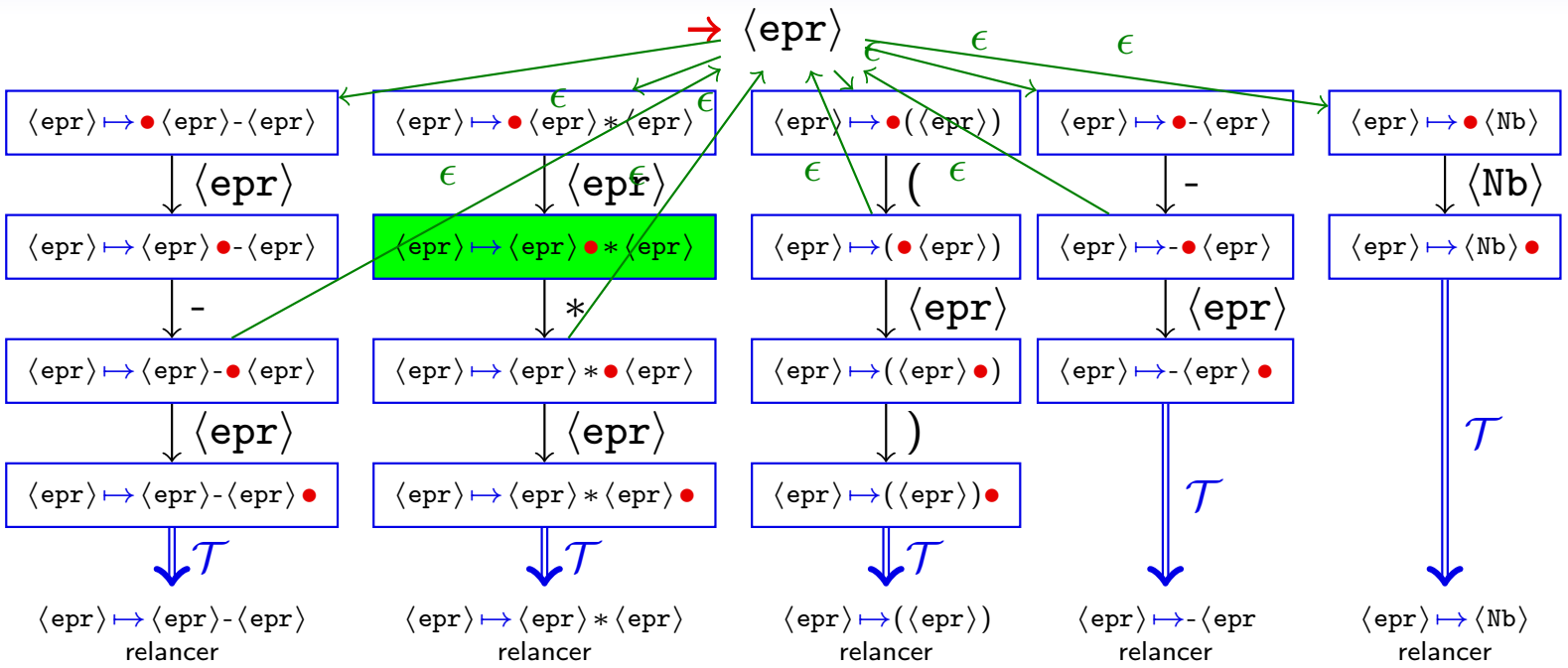


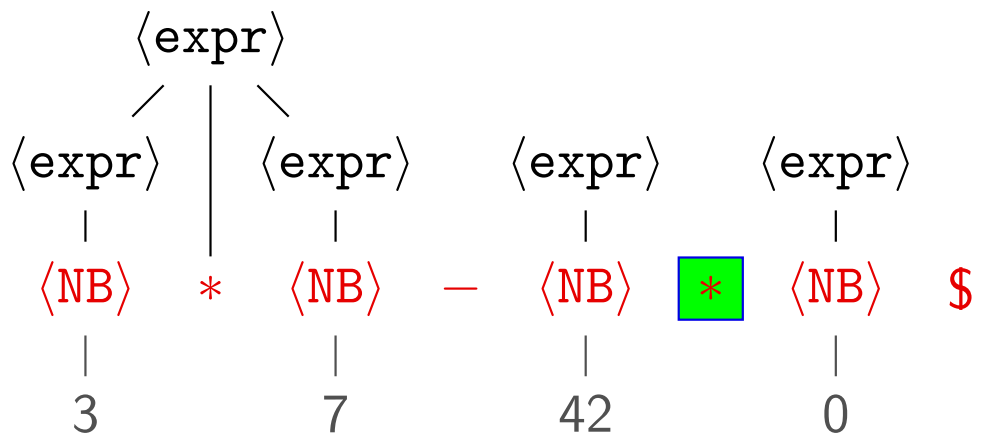
Second attempt in motion



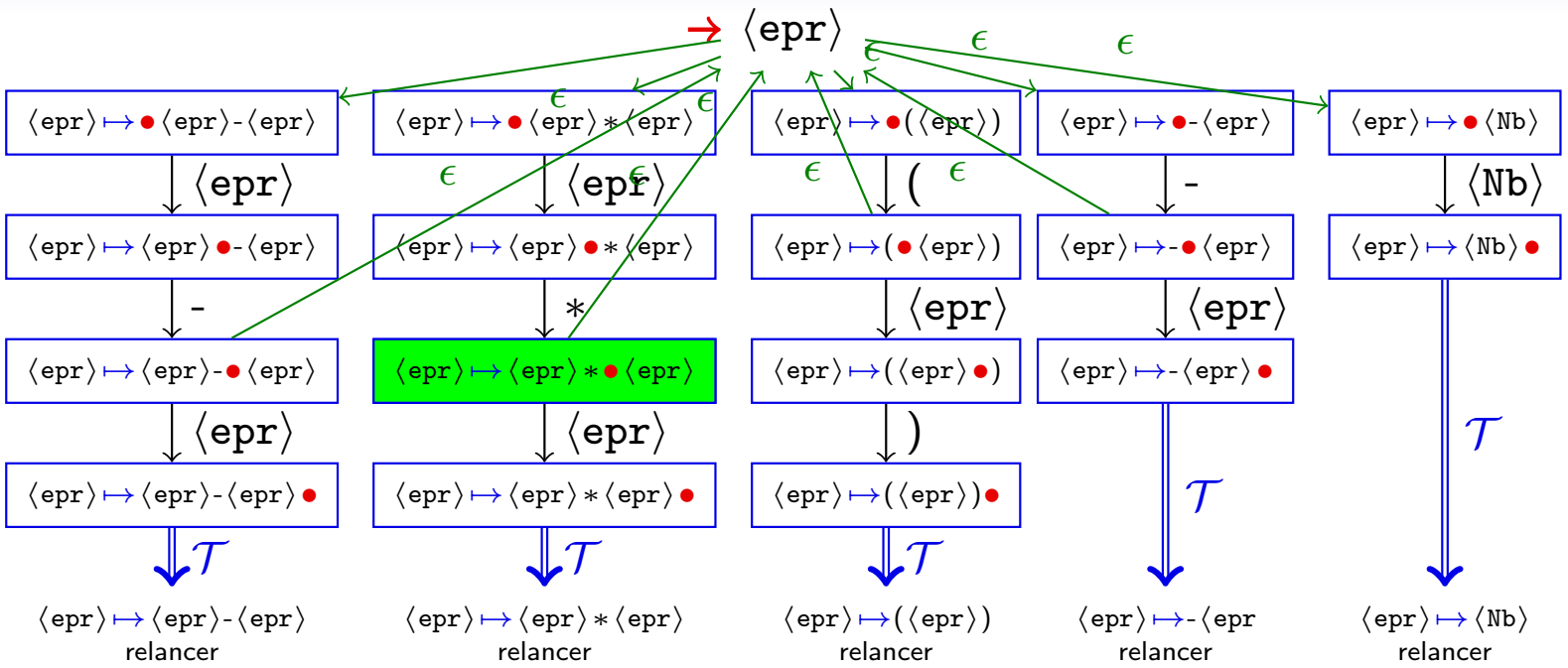


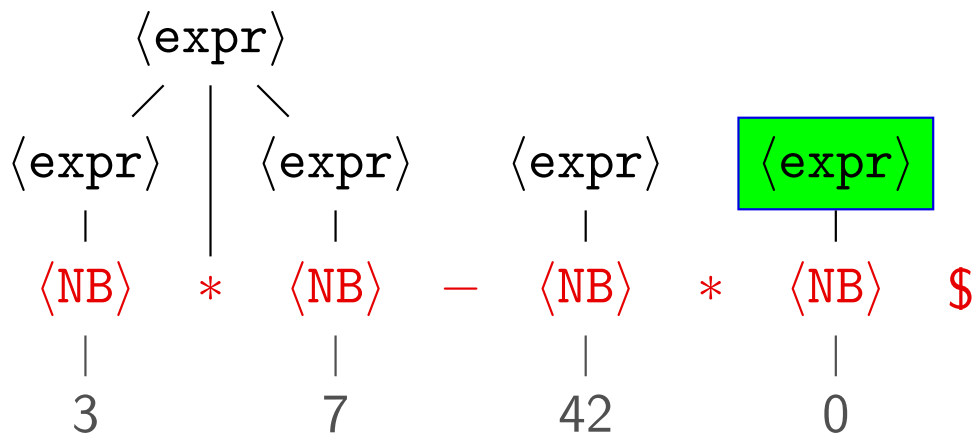
Second attempt in motion



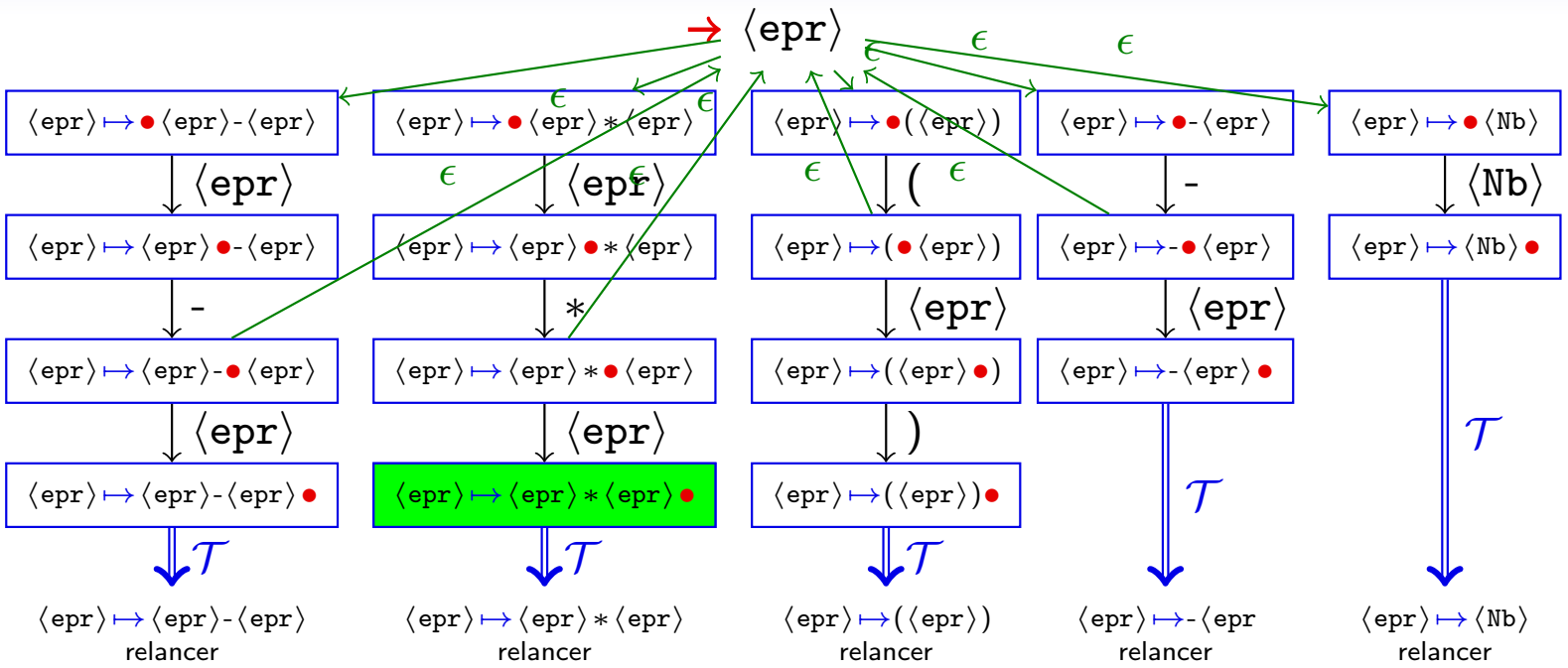


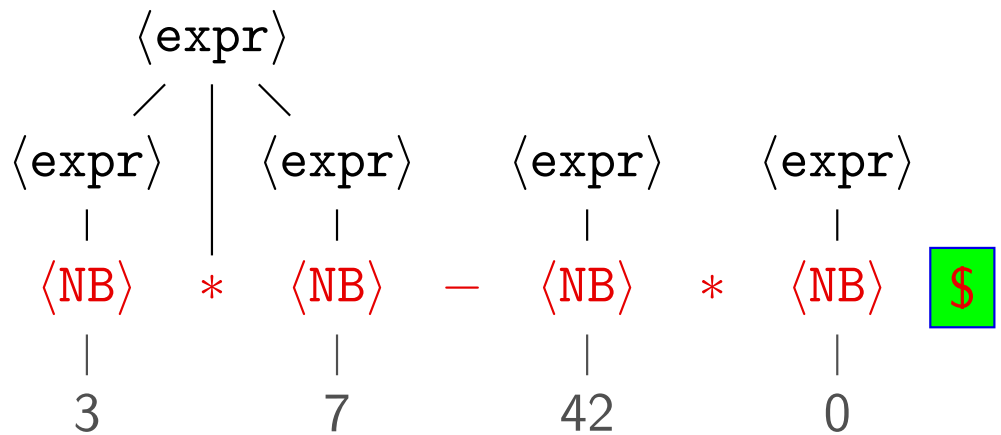
Second attempt in motion



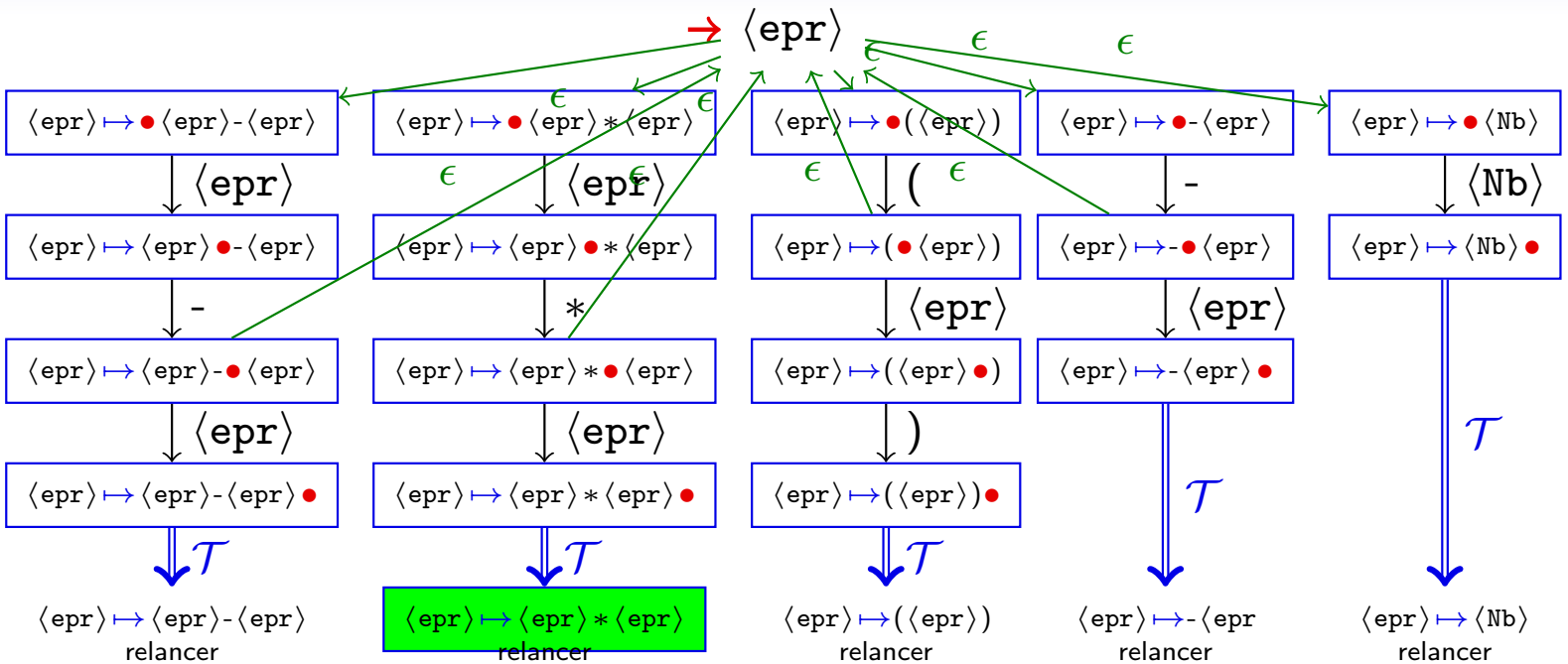


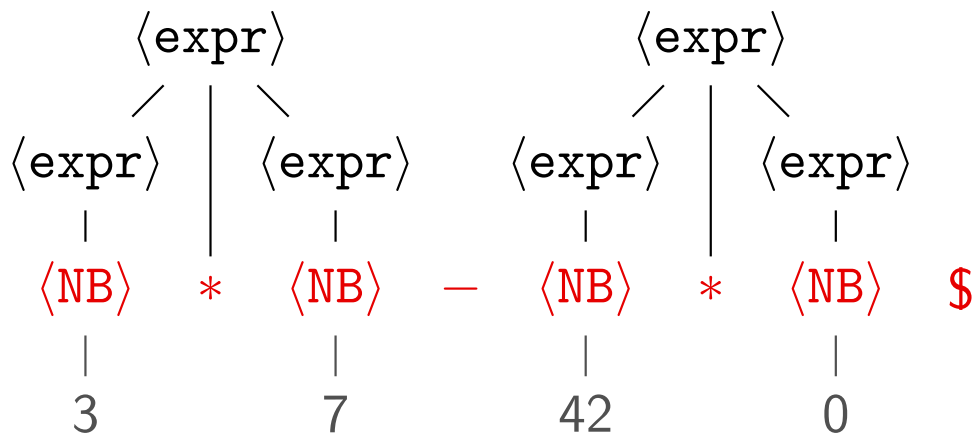
Second attempt in motion



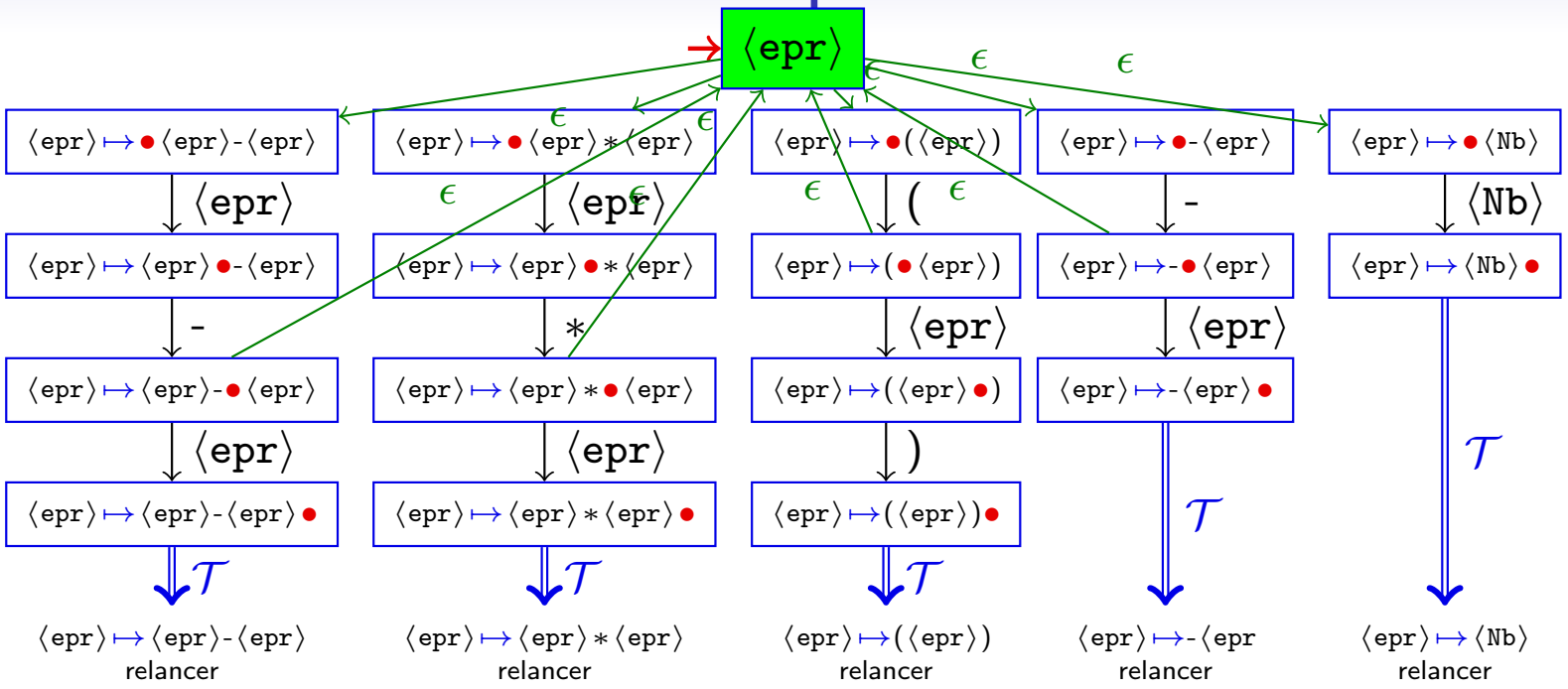


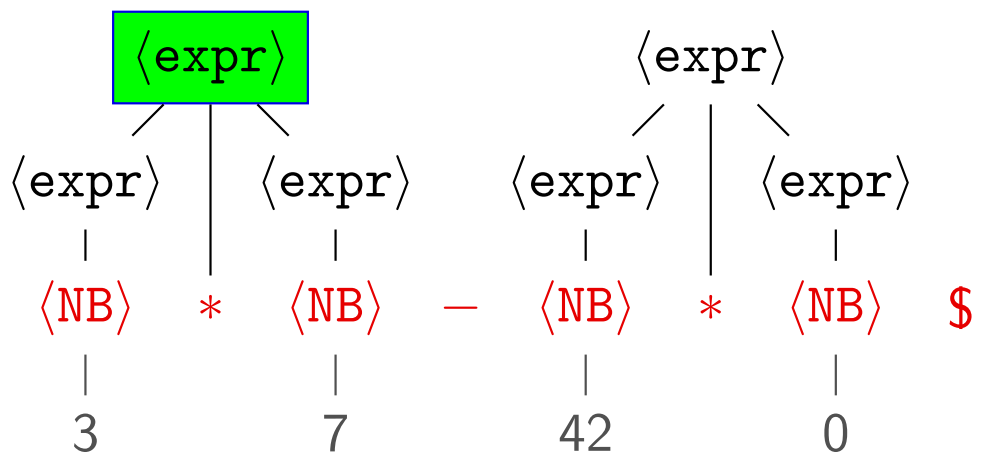
Second attempt in motion



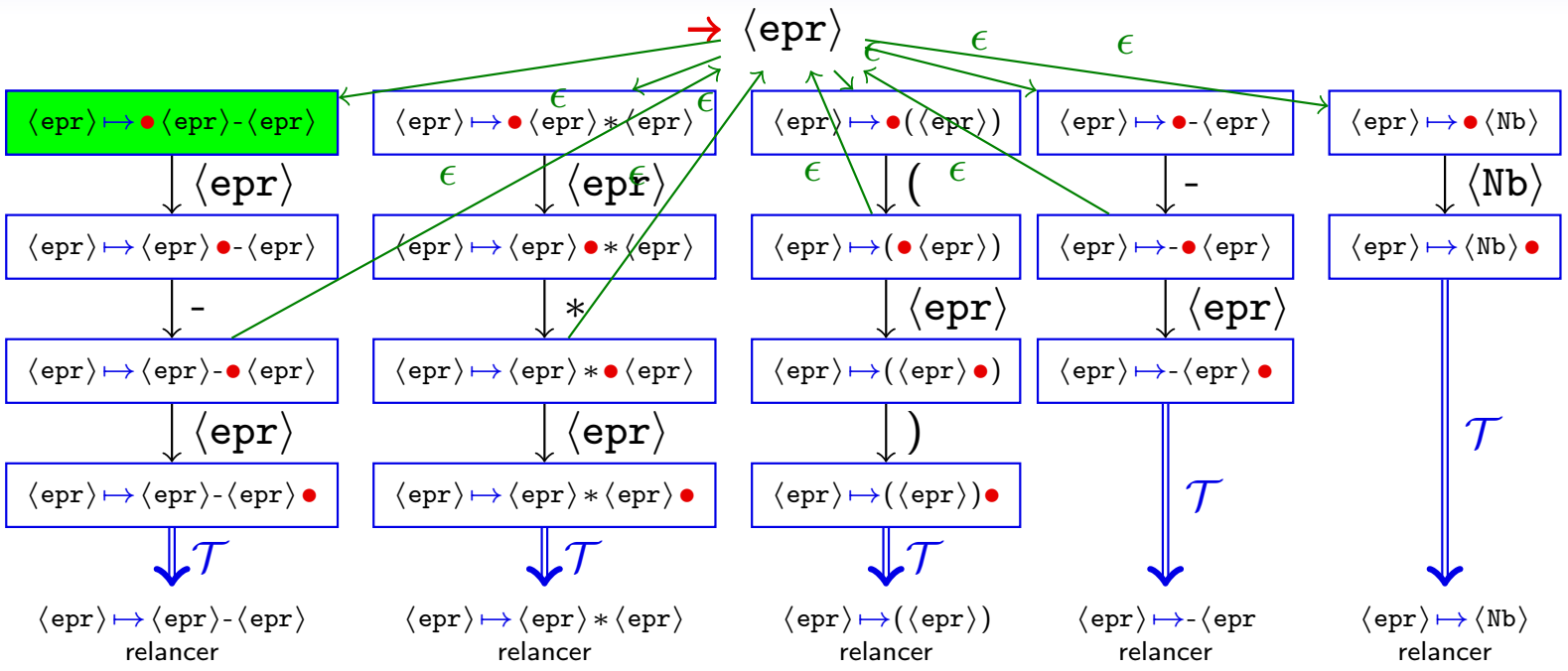


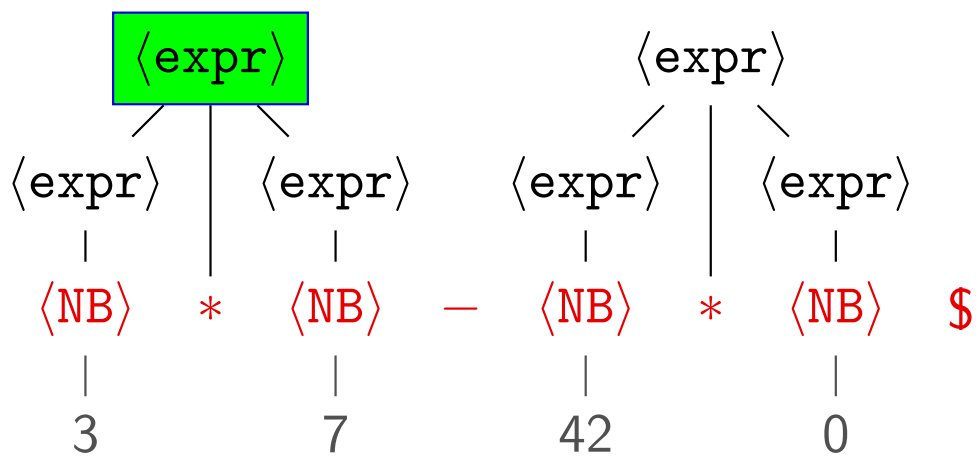
Second attempt in motion



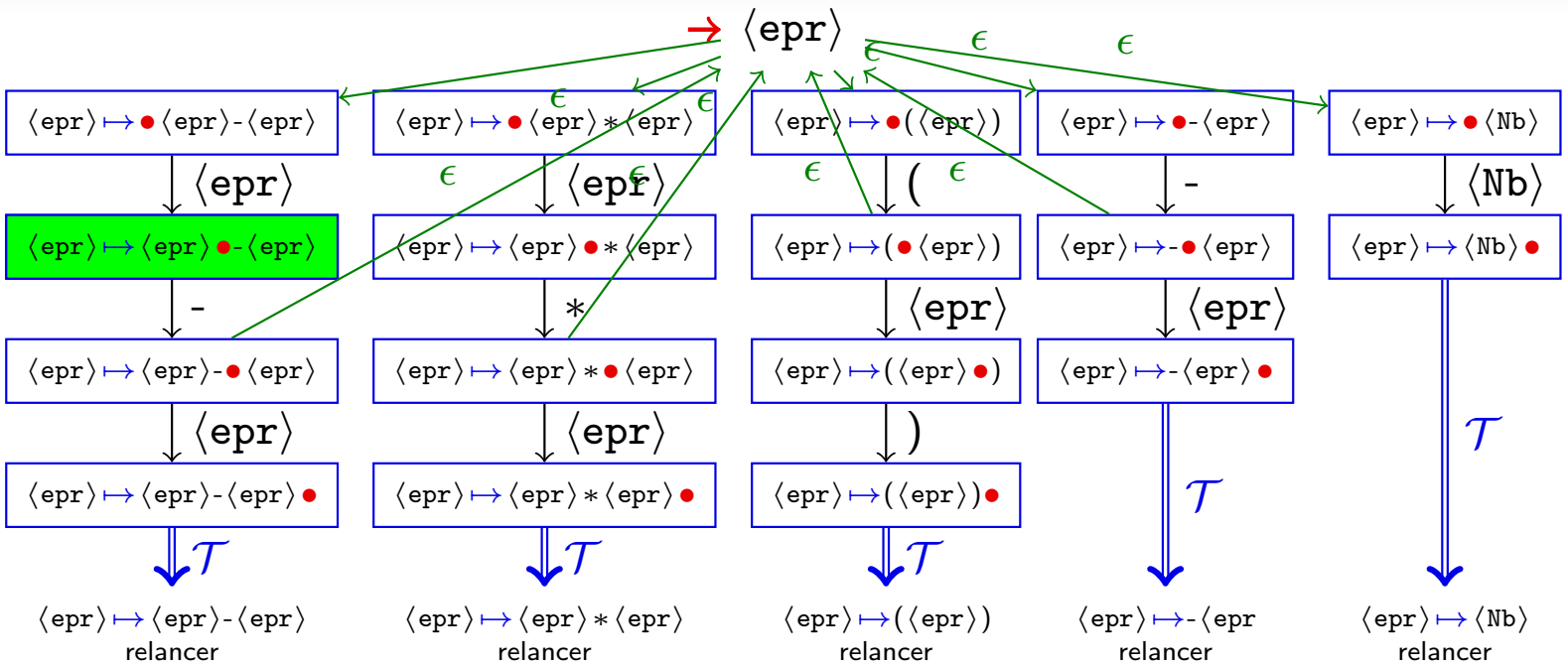


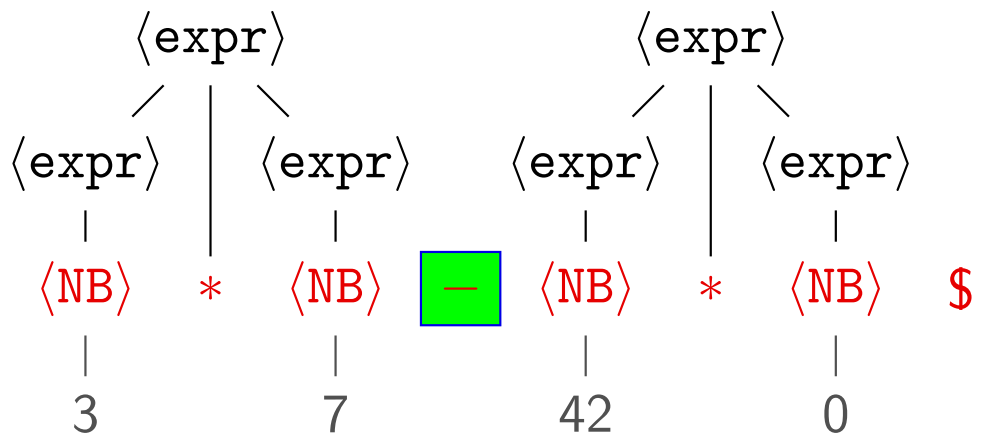
Second attempt in motion



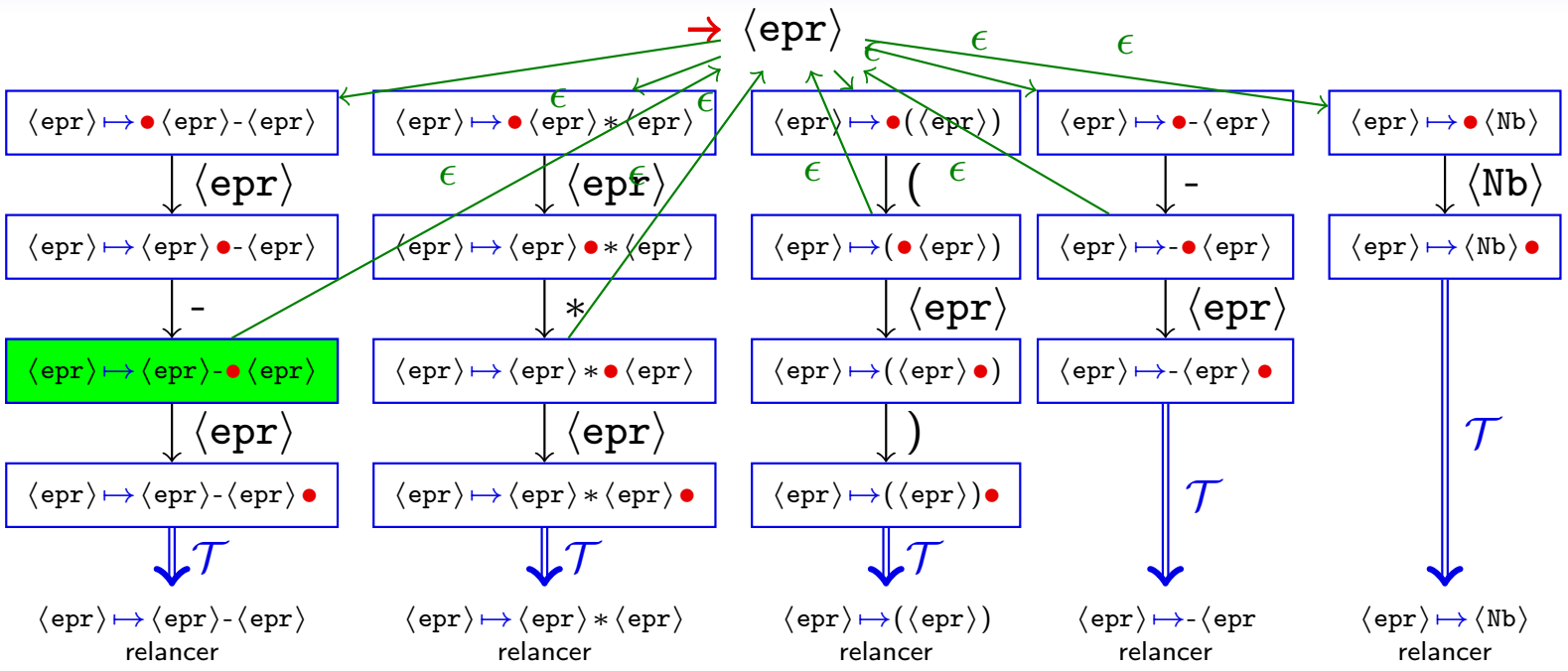


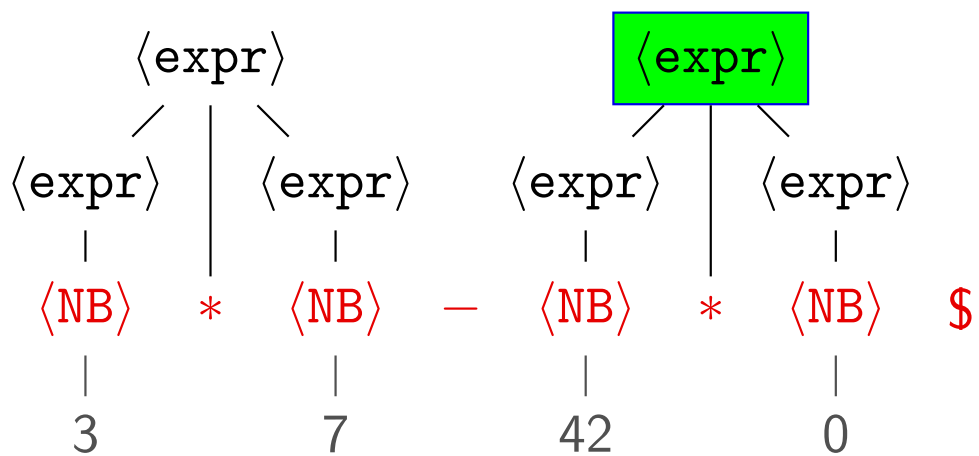
Second attempt in motion



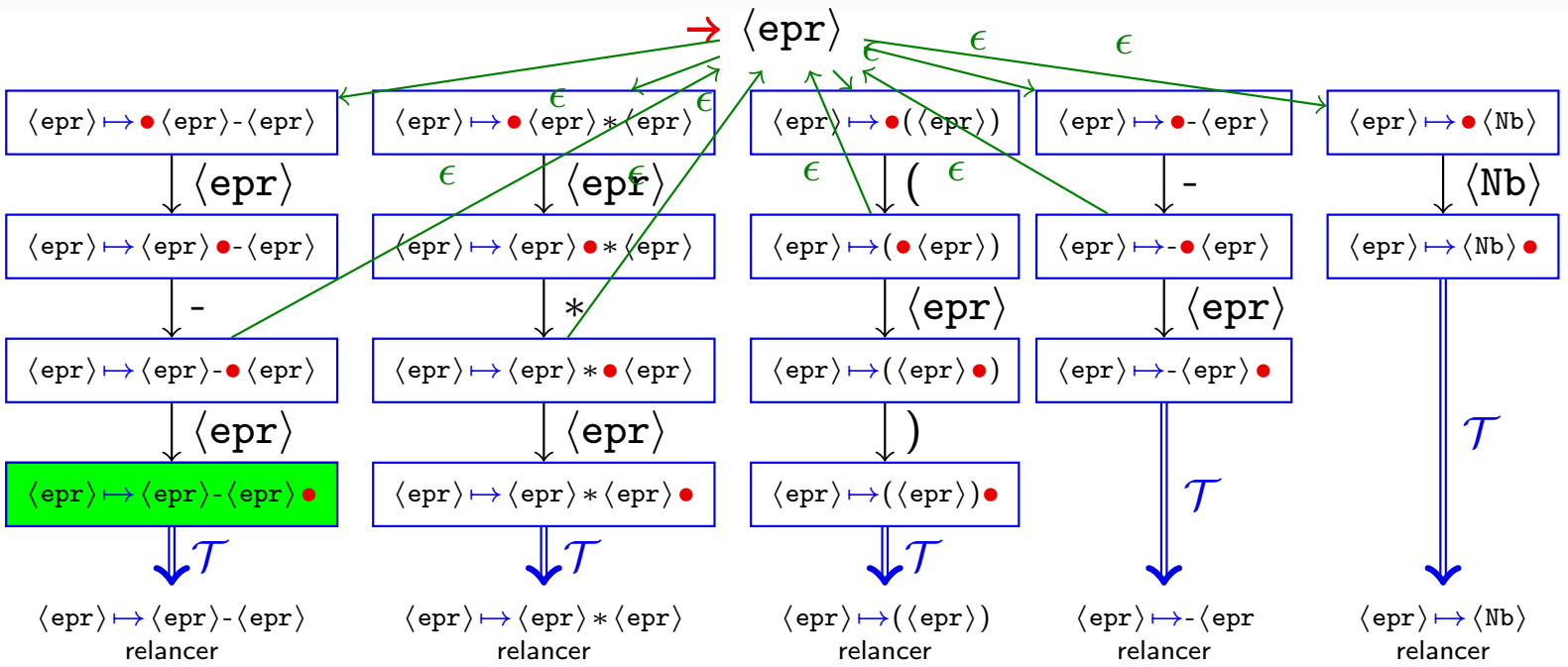


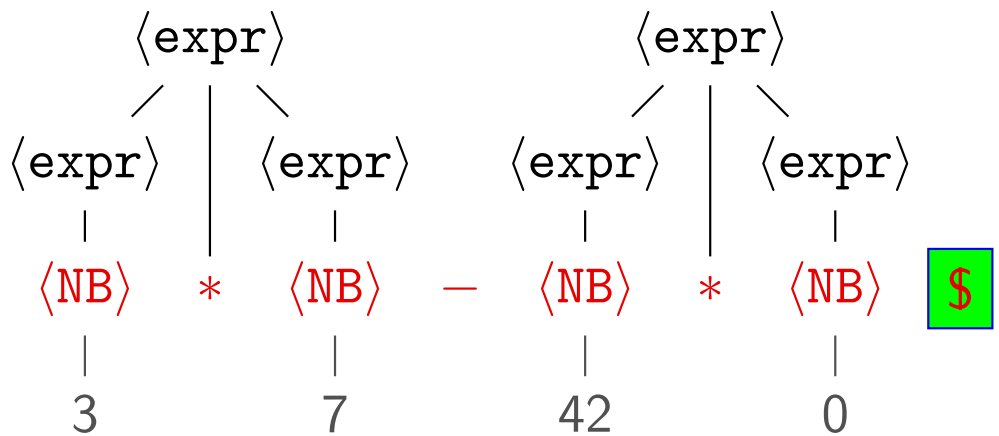
Second attempt in motion



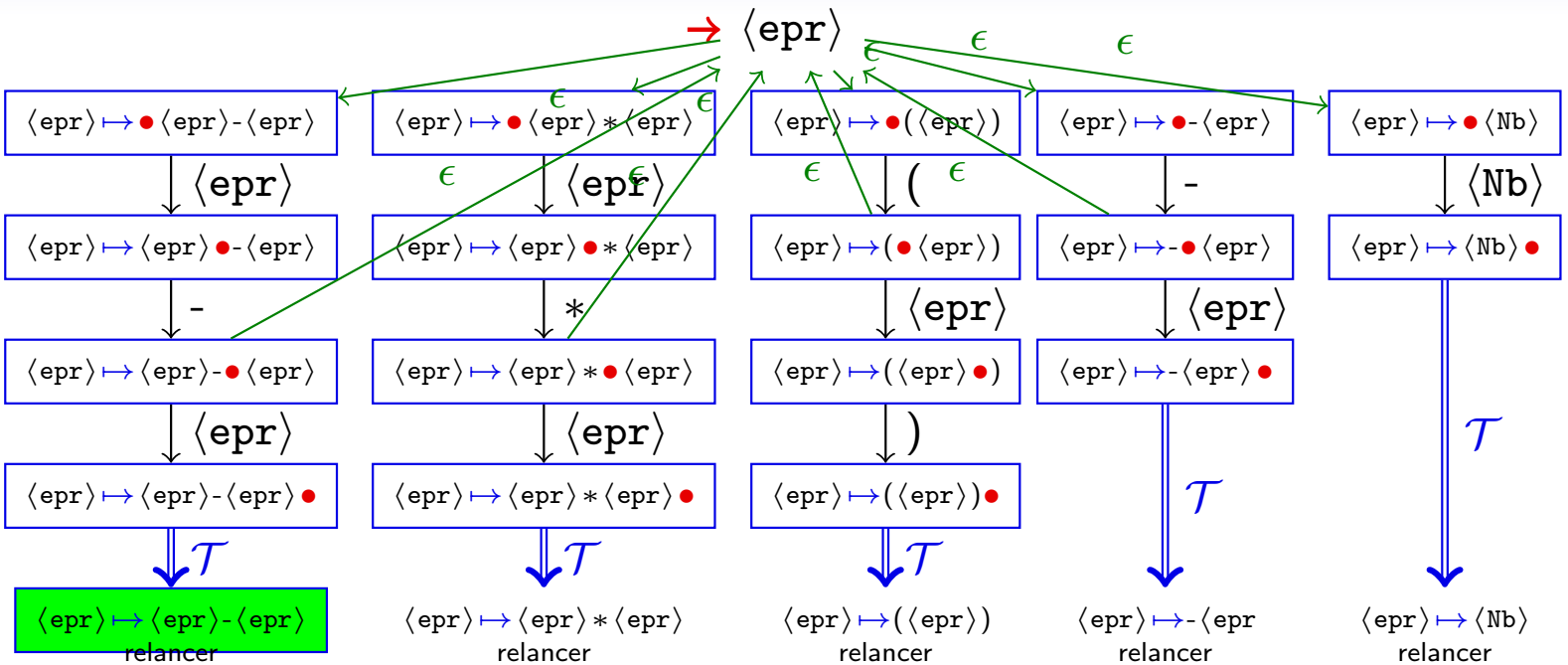


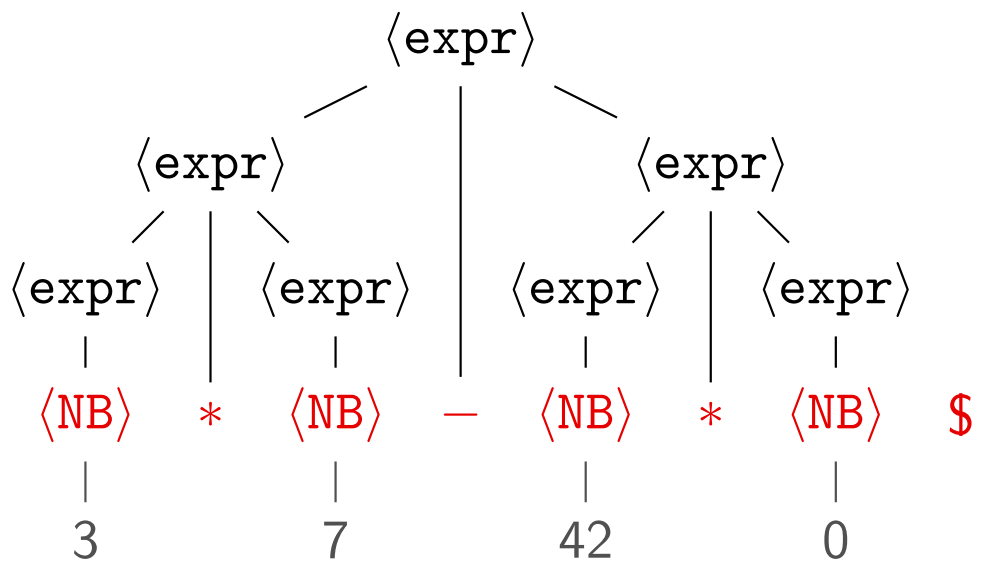
Second attempt in motion



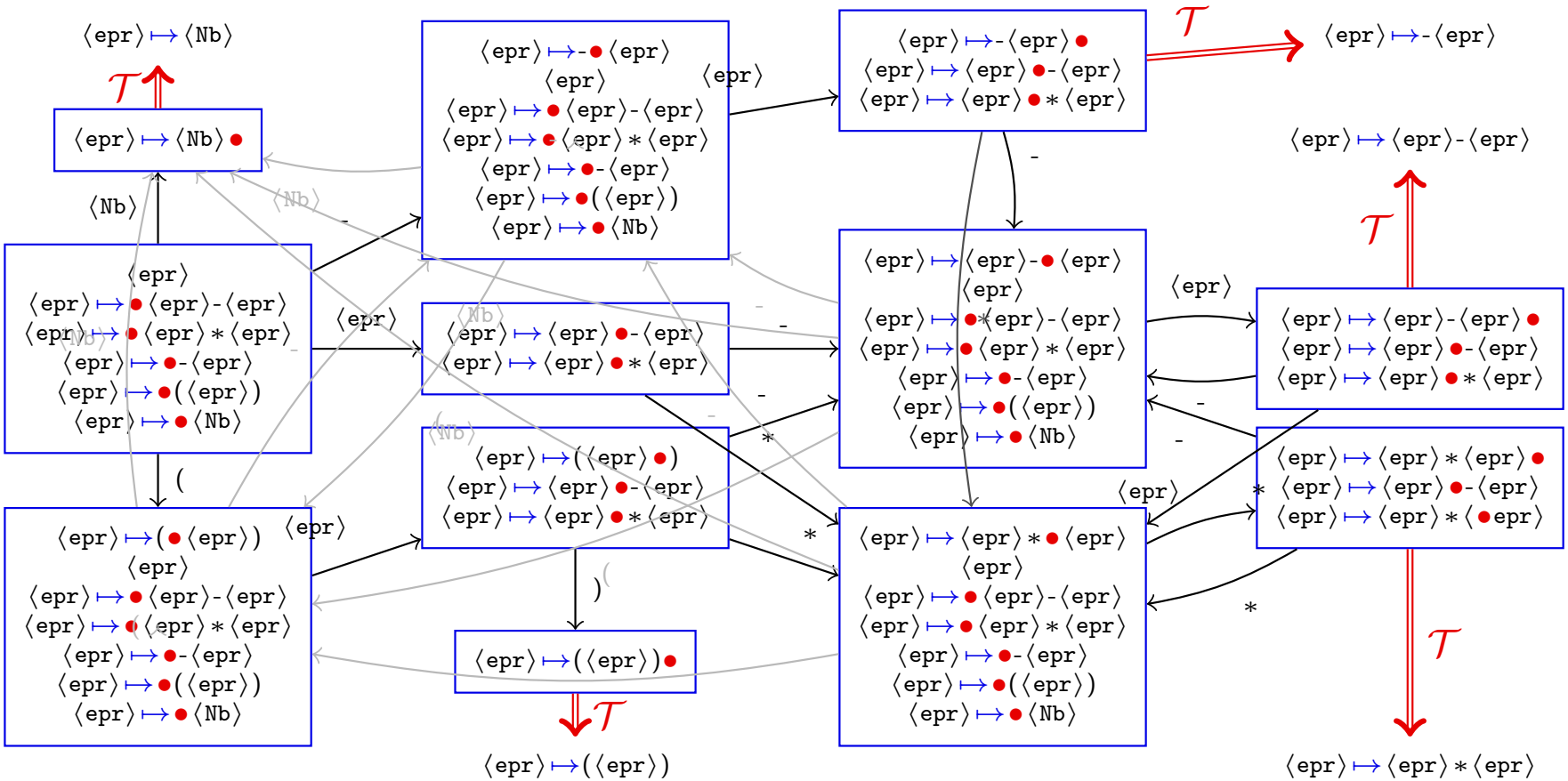


Second attempt in motion





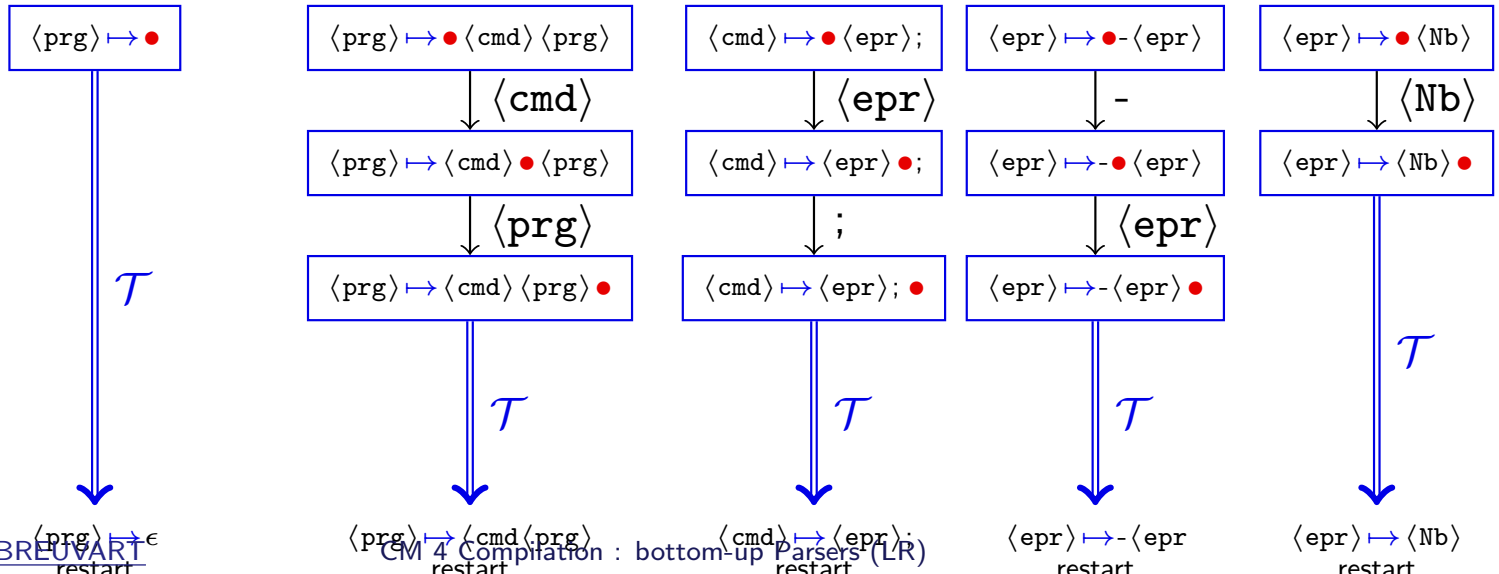
Pseudo-determinisation : LR_0 Parser



The Non-Deterministic Automaton

Let's try it again

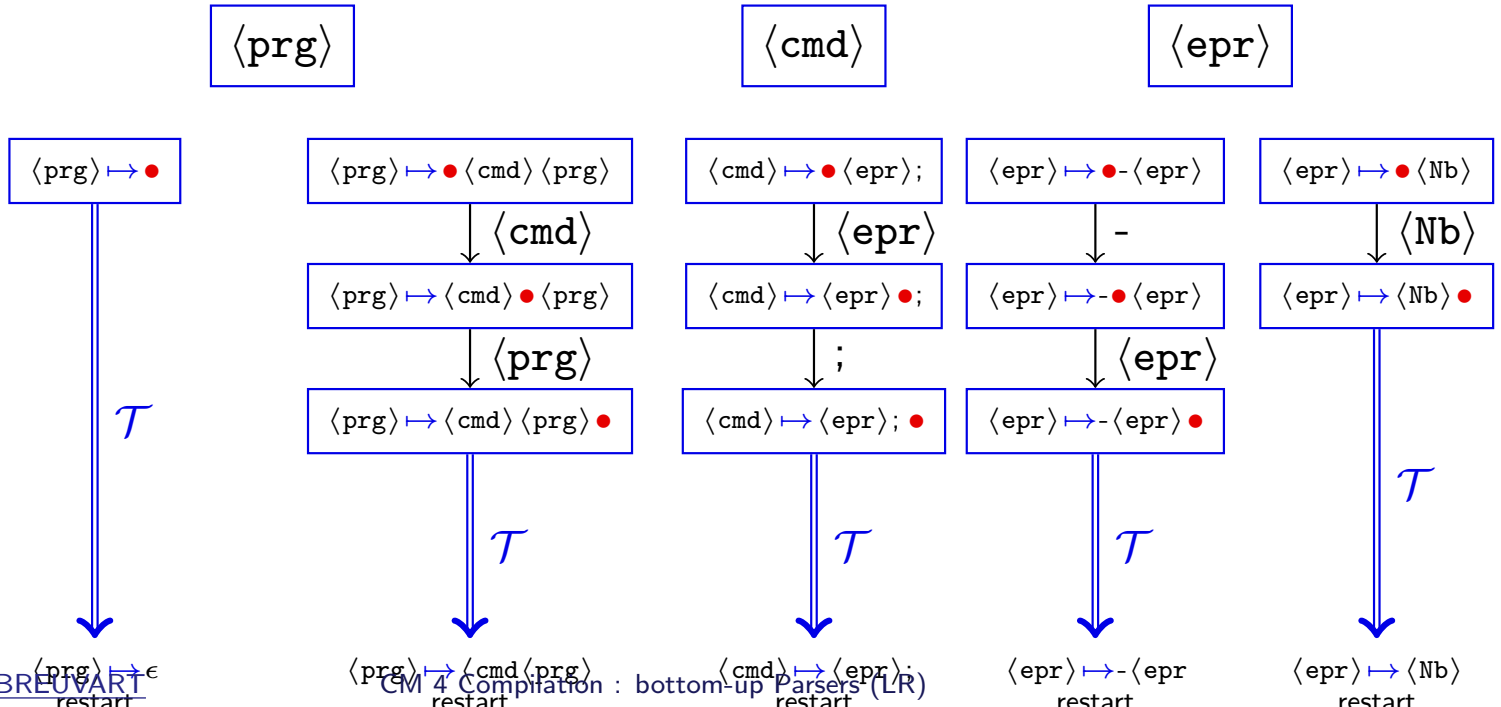
- shifts reading each rules,



The Non-Deterministic Automaton

Let's try it again

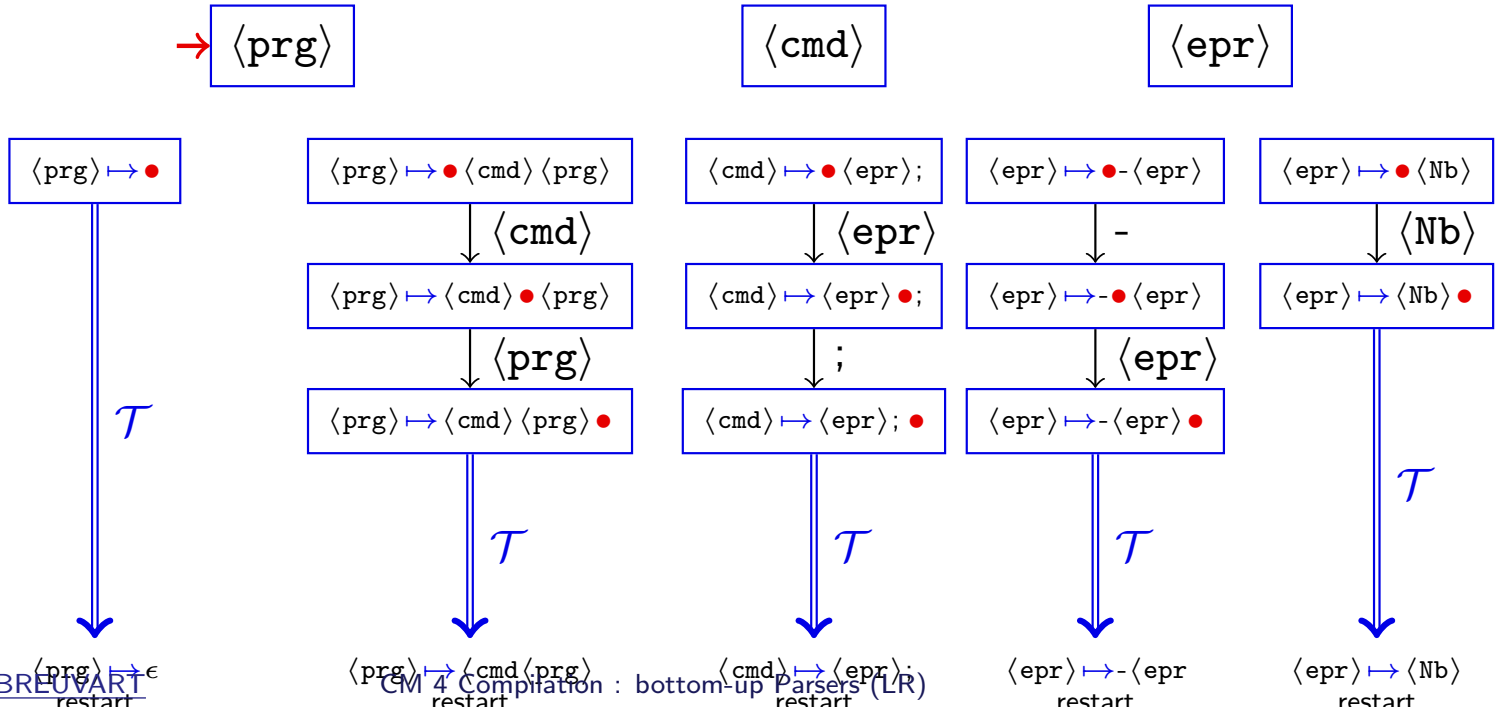
- shifts reading each rules,
- an additional state for each non-terminal N,



The Non-Deterministic Automaton

Let's try it again

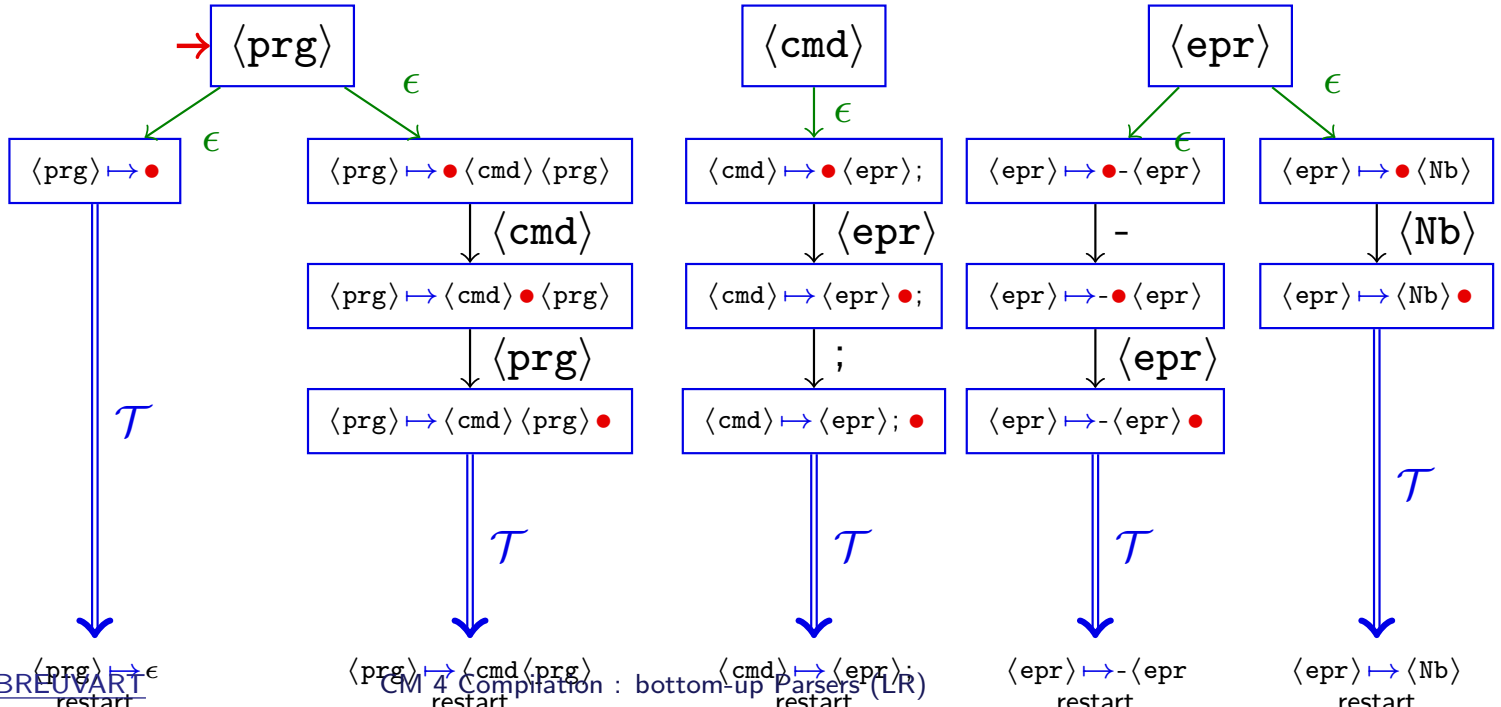
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,



The Non-Deterministic Automaton

Let's try it again

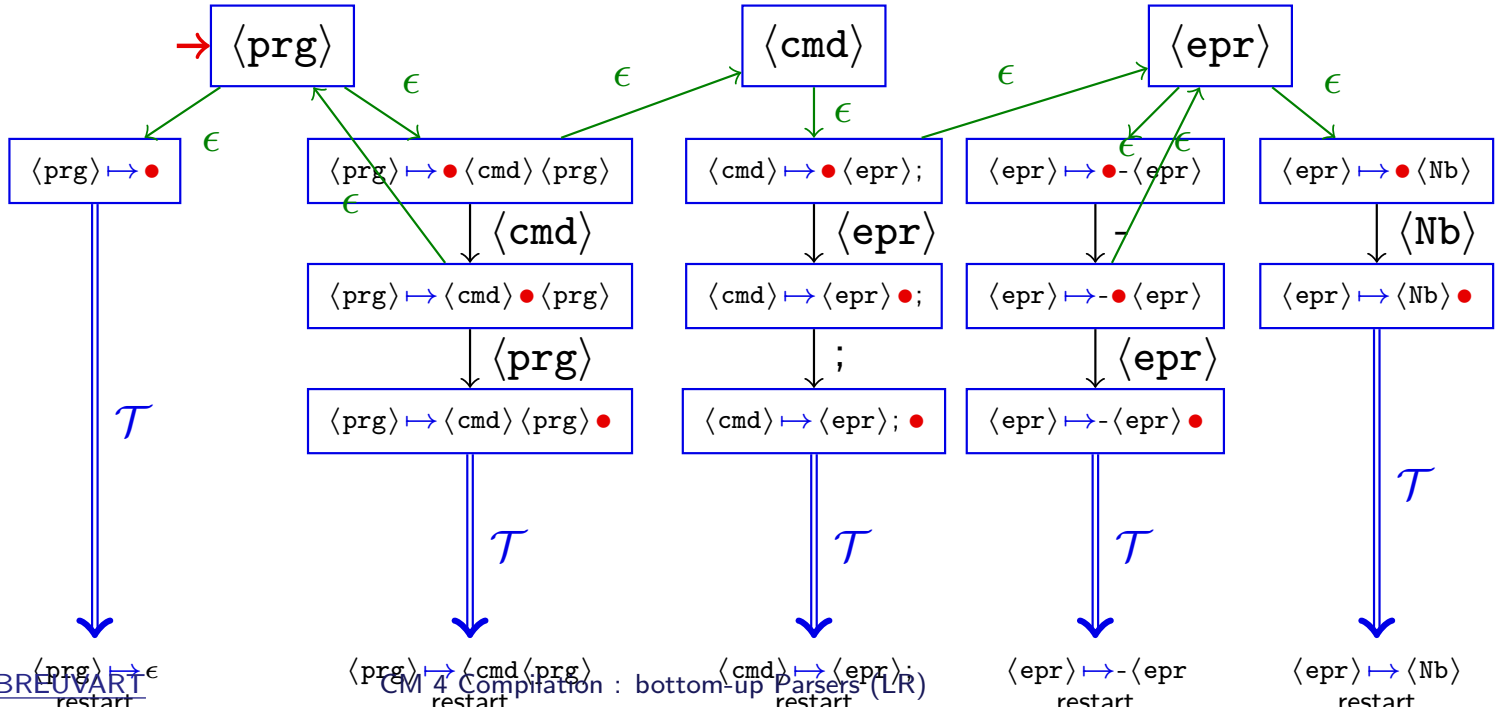
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,
- add ϵ-transitions toward each rule,



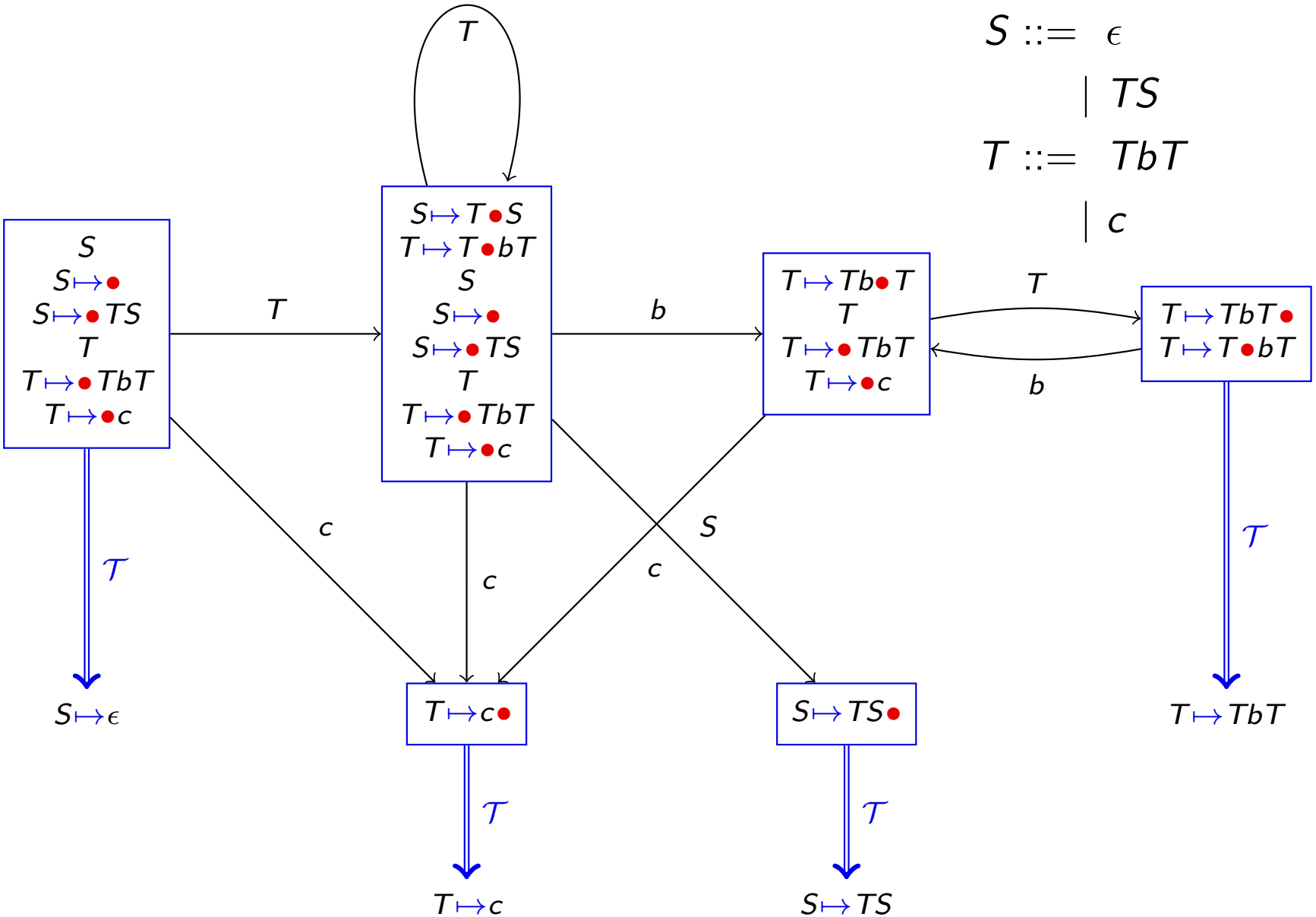
The Non-Deterministic Automaton

Let's try it again

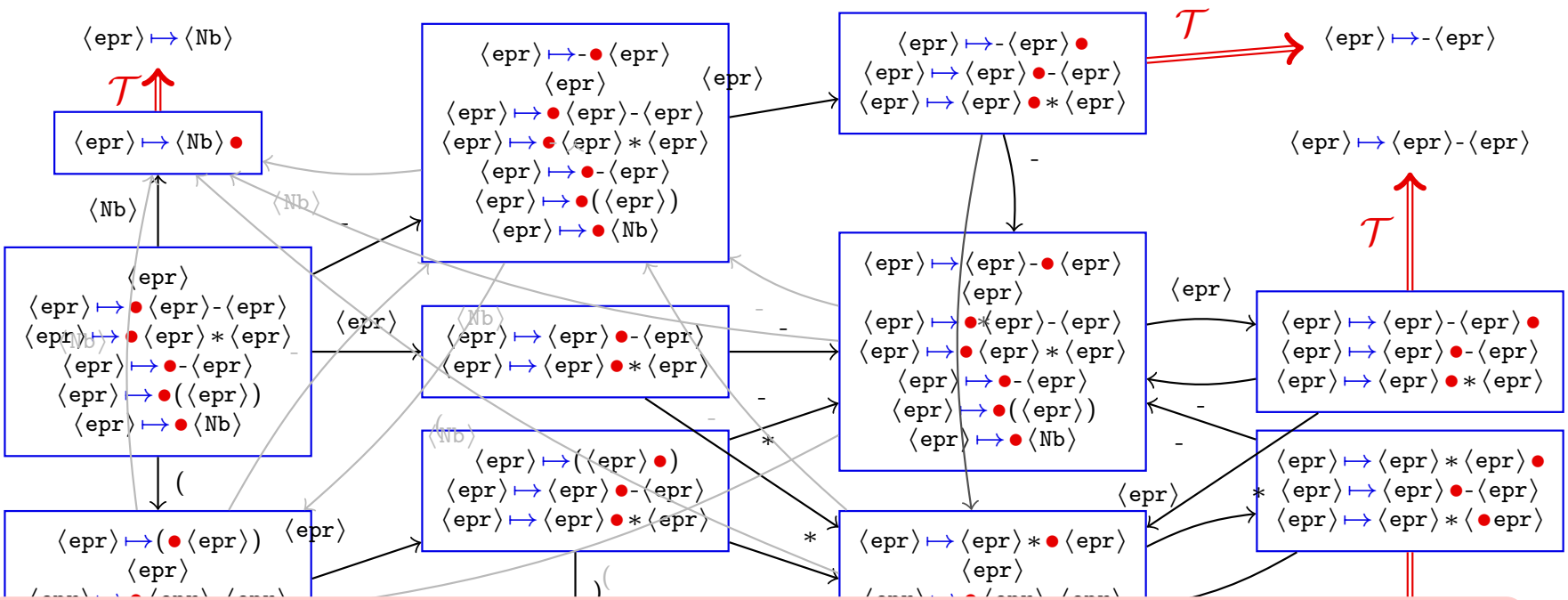
- shifts reading each rules,
- an additional state for each non-terminal N,
- of which the principal is initial,
- add ϵ-transitions toward each rule,
- for each state $M \mapsto w_1 \bullet N w_2$, add an ϵ-transition toward N



Pseudo-determinisation : Parser LR_0



Pseudo-determinisation : LR_0 Parser



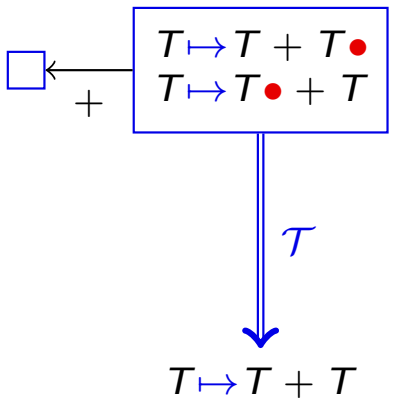
Conflict resolution

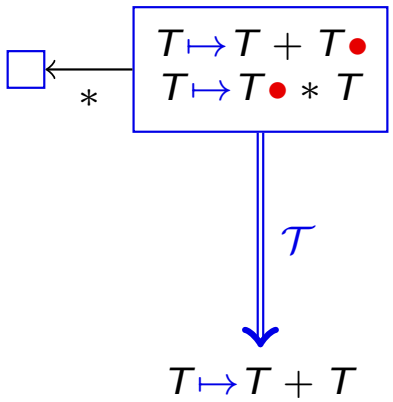
Remark: we will use the word "reduce" for "action" to match literature

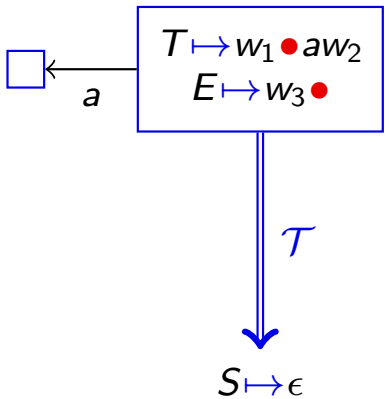
- Precedence : solve some shift-action and action-action conflicts
- Associativity : solve other shift-action conflicts
- But not allways sufficient to solve conflicts...

A grammar which LR_0 parser is deterministic is called a LR_0 grammar

Some usual conflicts







Associativity

right associativity :
↳ choose the shift

left associativity :
↳ choose the action

Precedence

$*$ precedence :
↳ choose the shift

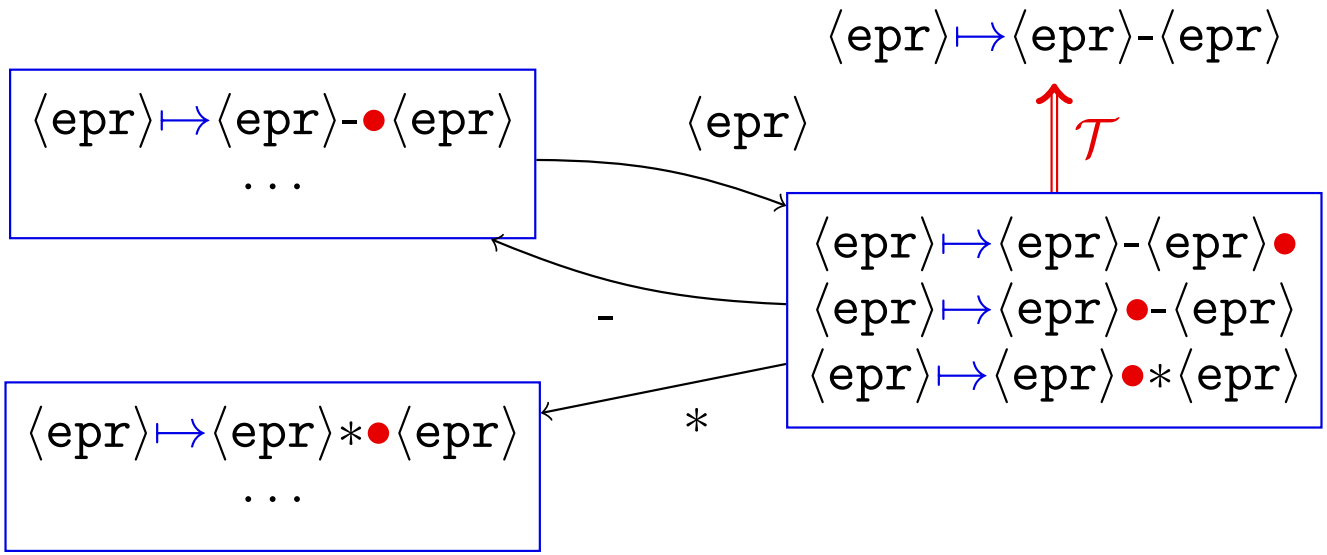
$+$ precedence :
↳ choose the action

Not arrising from ambiguity

eg, when no a after S

Need of stronger algo
↳ SLR, LALR, LR1 ...

Conflicts solved by precedence and associativity



Shift/reduce from precedence

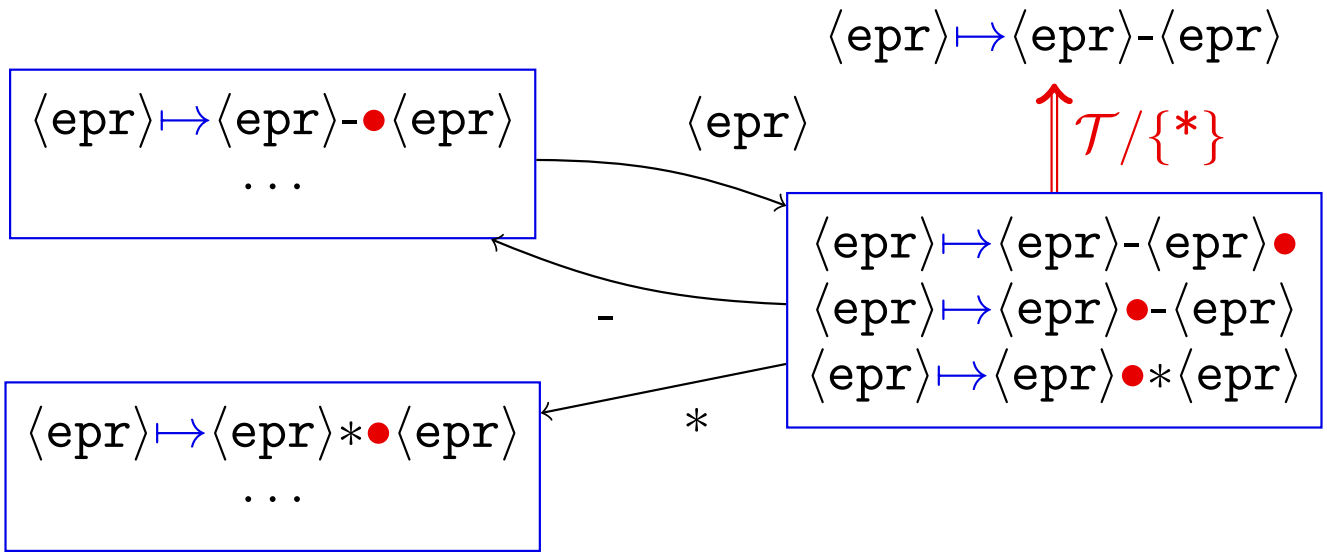
$\langle \texttt{epr} \rangle \mapsto \langle \texttt{epr} \rangle \bullet * \langle \texttt{epr} \rangle$

vs

$\langle \texttt{epr} \rangle \mapsto \langle \texttt{epr} \rangle - \langle \texttt{epr} \rangle \bullet$

Choose the prioritized rule

Conflicts solved by precedence and associativity



Shift/reduce from precedence

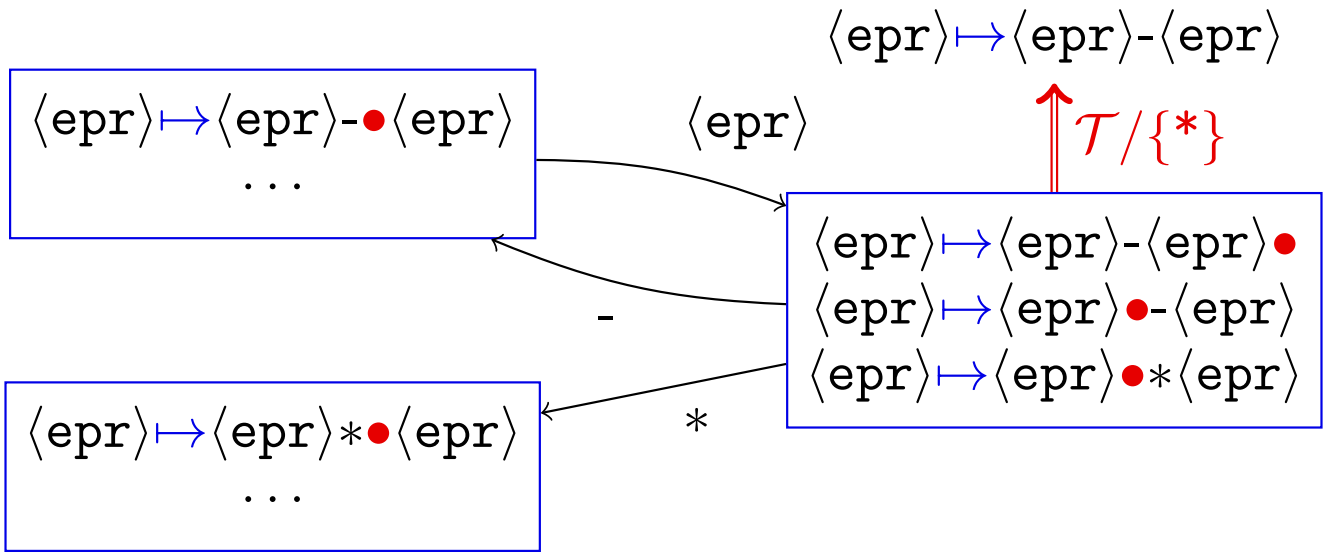
⟨epr⟩↦⟨epr⟩•*⟨epr⟩

vs

⟨epr⟩↦⟨epr⟩-⟨epr⟩•

Choose the prioritized rule

Conflicts solved by precedence and associativity



Shift/reduce from precedence

$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle \bullet * \langle \text{epr} \rangle$

vs

$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle - \langle \text{epr} \rangle \bullet$

Choose the prioritized rule

Shift/reduce from associativity

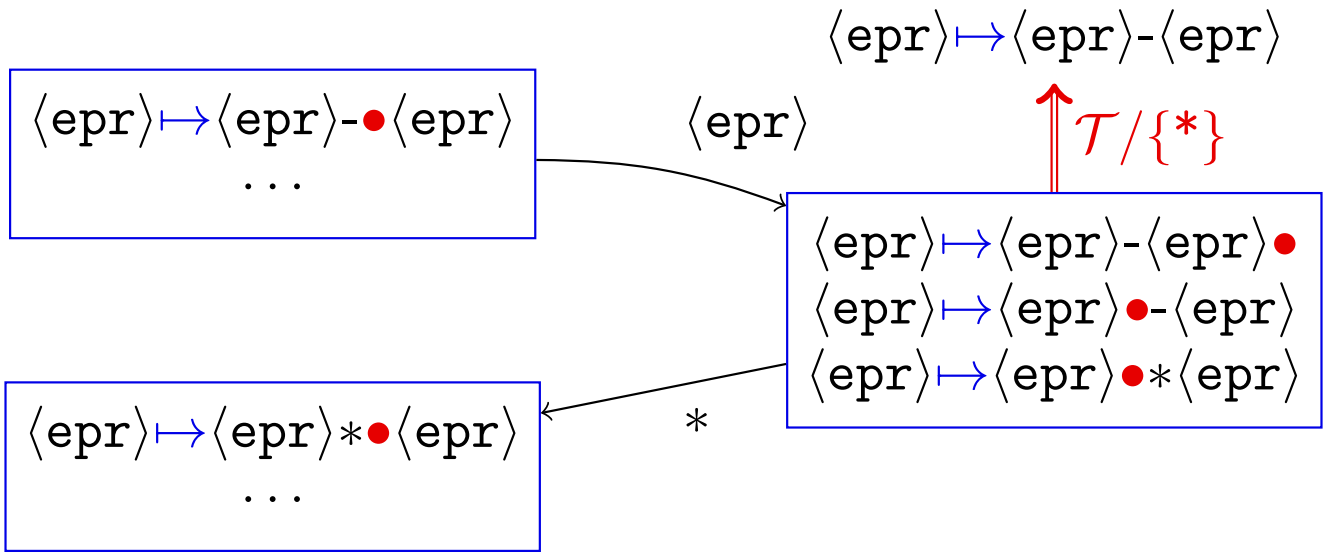
$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle \bullet - \langle \text{epr} \rangle$

vs

$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle - \langle \text{epr} \rangle \bullet$

Choose the shift if right asso.
Choose reduce reduce if right asso.

Conflicts solved by precedence and associativity



Shift/reduce from precedence

$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle \bullet * \langle \text{epr} \rangle$

vs

$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle - \langle \text{epr} \rangle \bullet$

Choose the prioritized rule

Shift/reduce from associativity

$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle \bullet - \langle \text{epr} \rangle$

vs

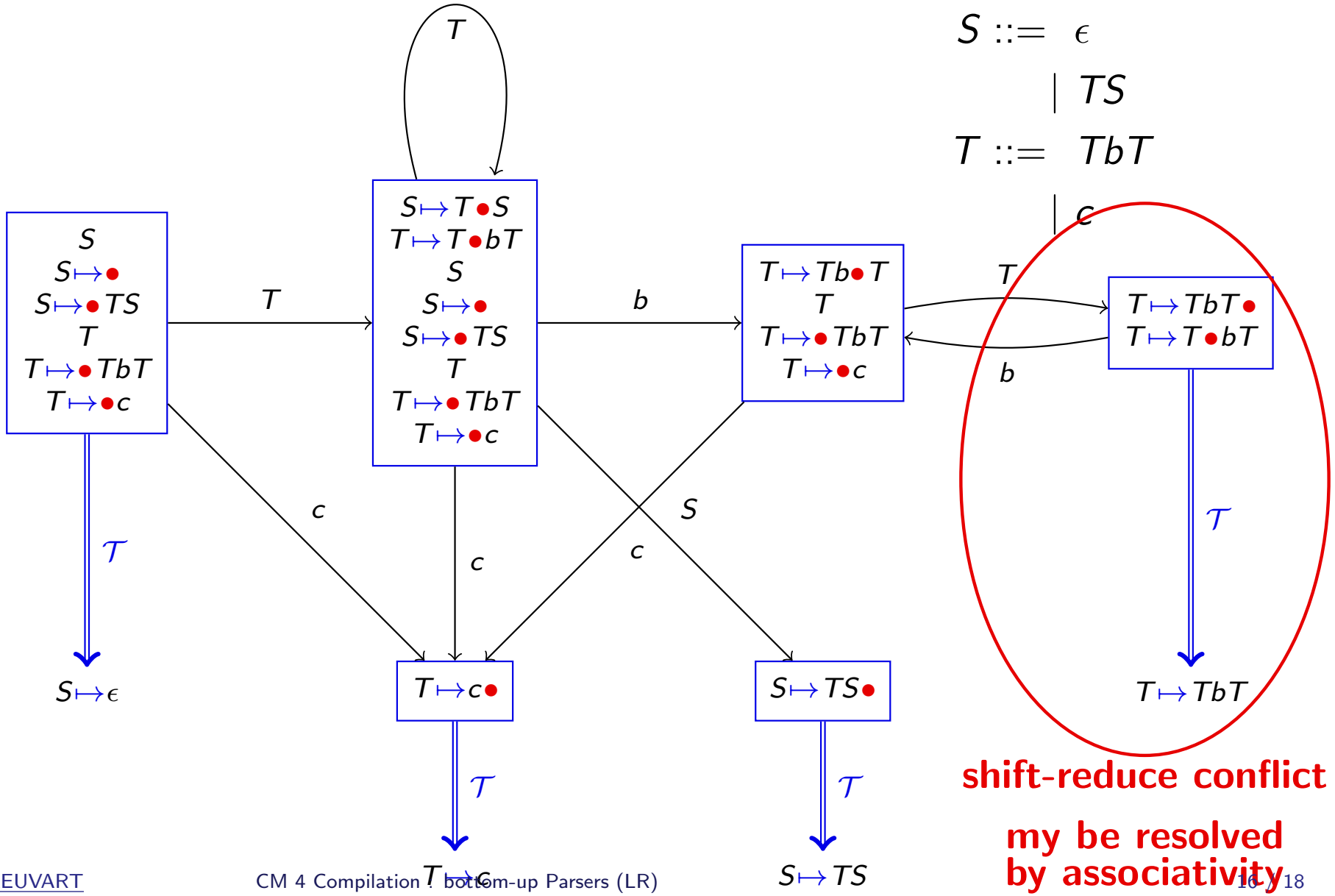
$\langle \text{epr} \rangle \mapsto \langle \text{epr} \rangle - \langle \text{epr} \rangle \bullet$

Choose the shift if right asso.
Choose reduce reduce if right asso.

Warning: decision-making differs drastically from the lexer's.

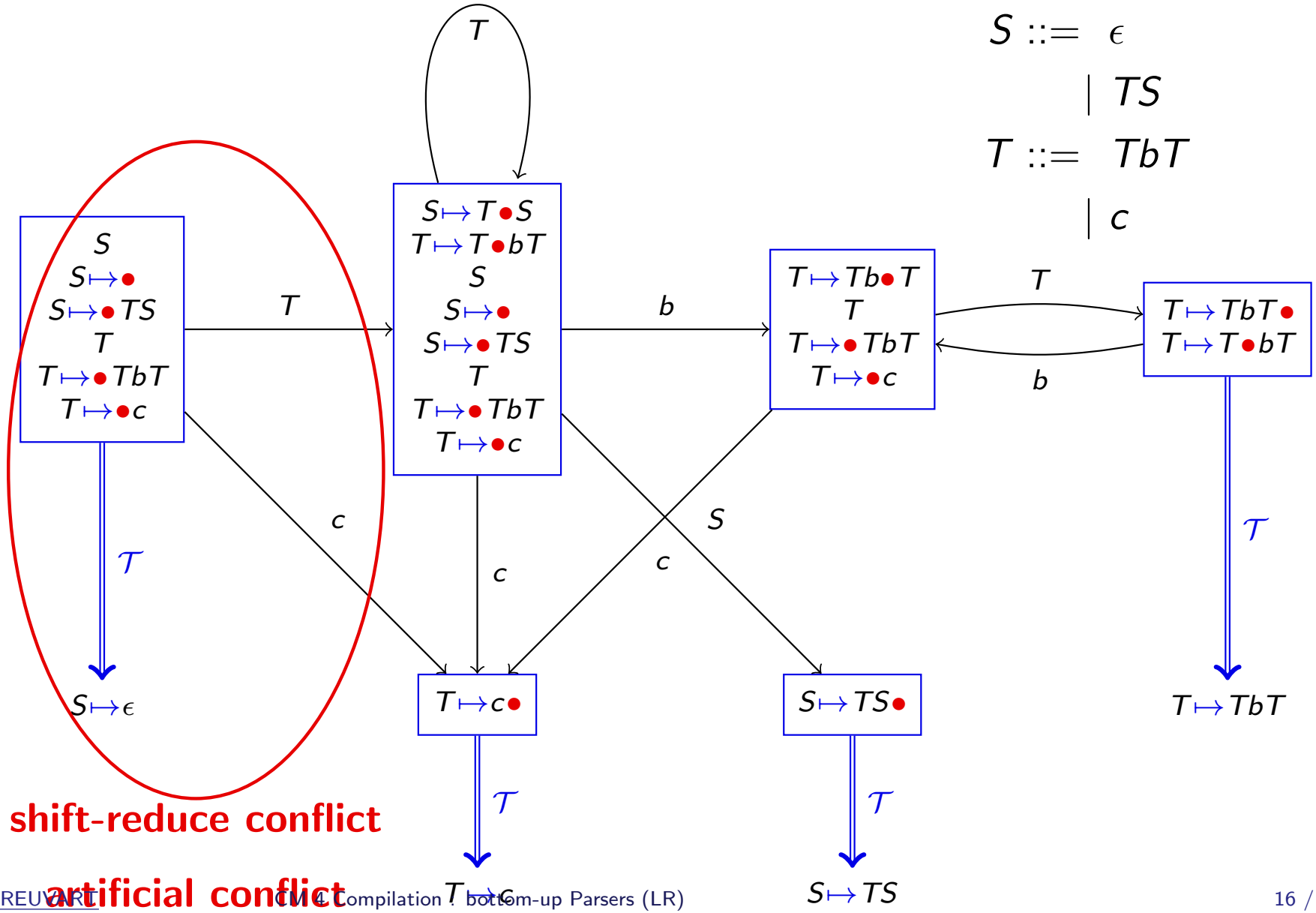
Warning, "non-ambiguous" conflicts may remain

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



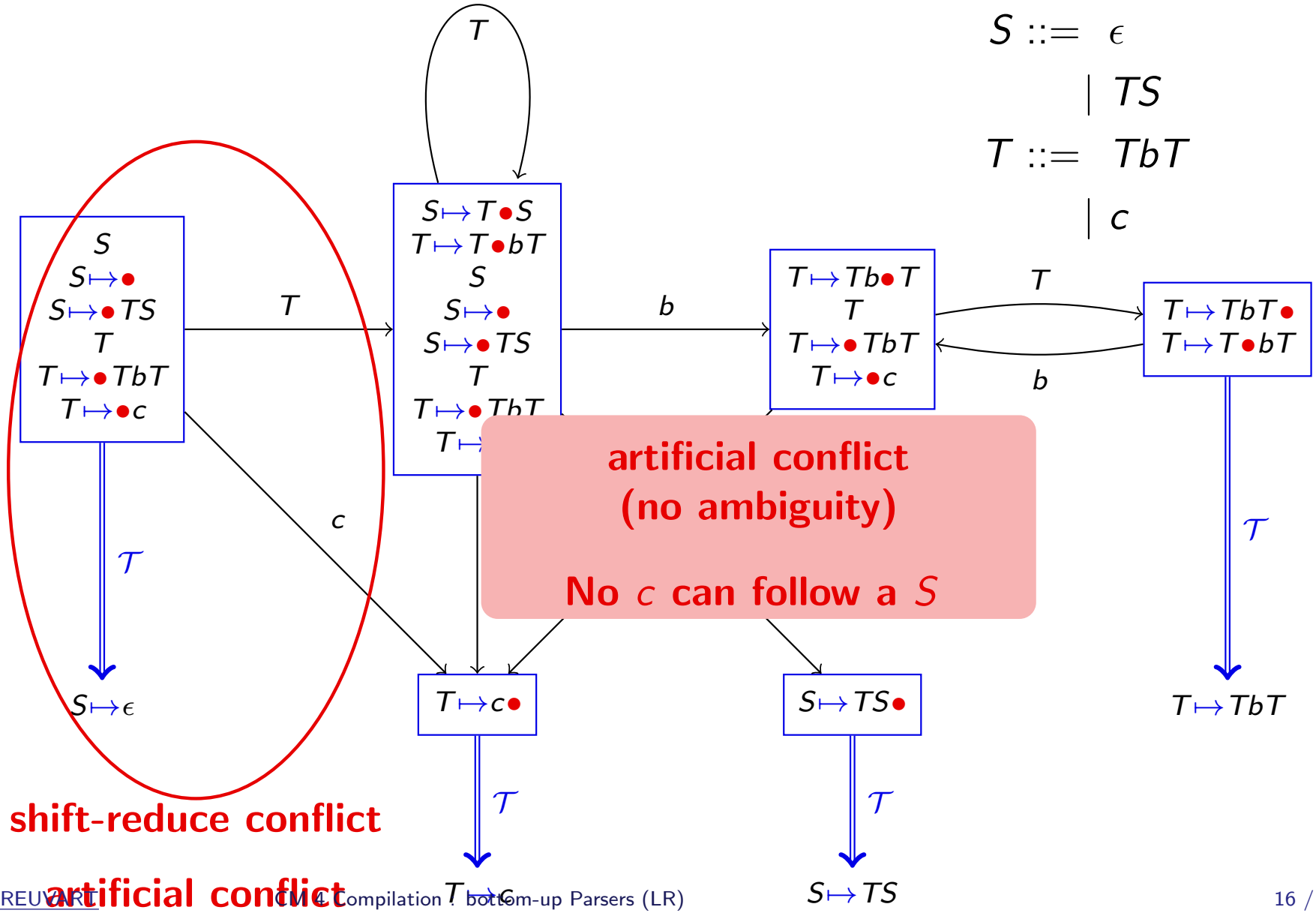
Warning, "non-ambiguous" conflicts may remain

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



Warning, "non-ambiguous" conflicts may remain

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



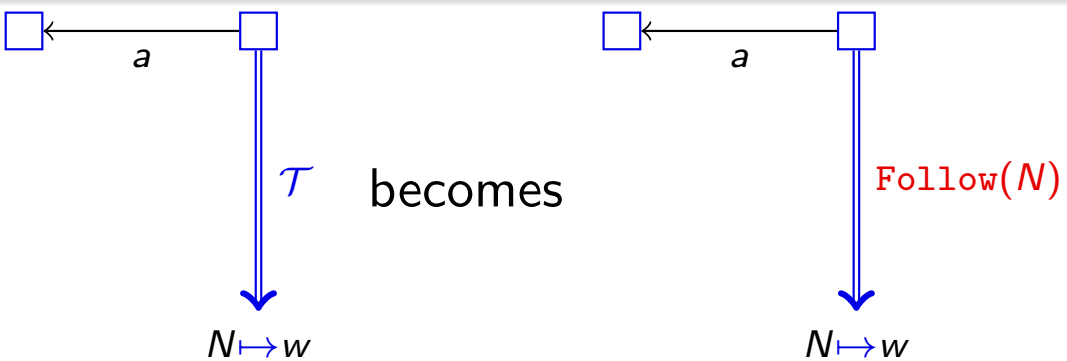
SLR Parser: narrowing conditions of actions

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

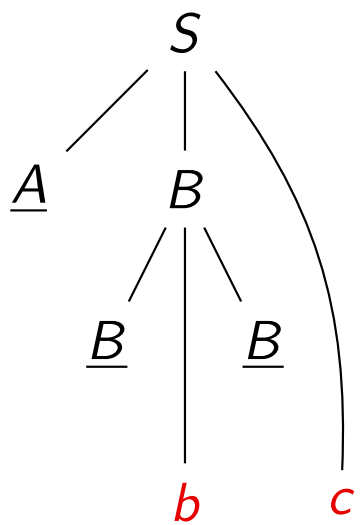
Prior to "follows", let's look at easier "firsts"

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

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



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

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

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

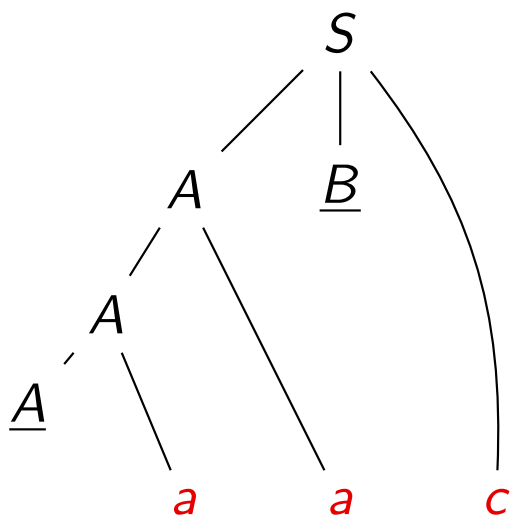
$c \in \texttt{Follow}_G(N) \iff$ { There exists a ST G with an internal node N which leftmost terminal leas RIGHT OF N is c.

Example ins

$S ::= ABc$

$A ::= \epsilon \mid aA$

$B ::= \epsilon \mid BbB$



Thus

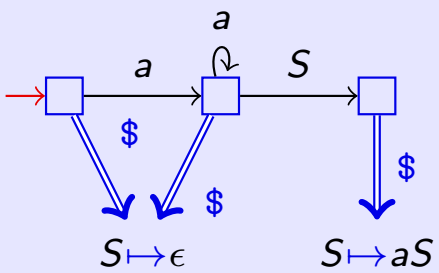
$a \in \texttt{Follow}(A)$

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

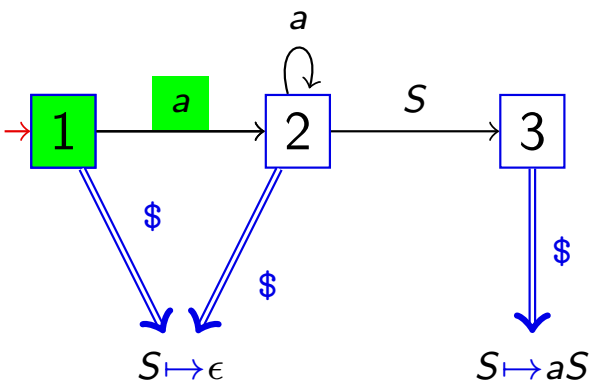
Optimization : linear time

The constructed SLR automaton can be of quadratic complexity

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

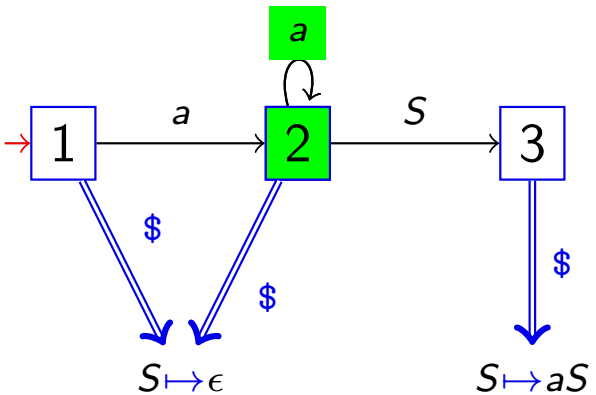


Example of non-linear execution



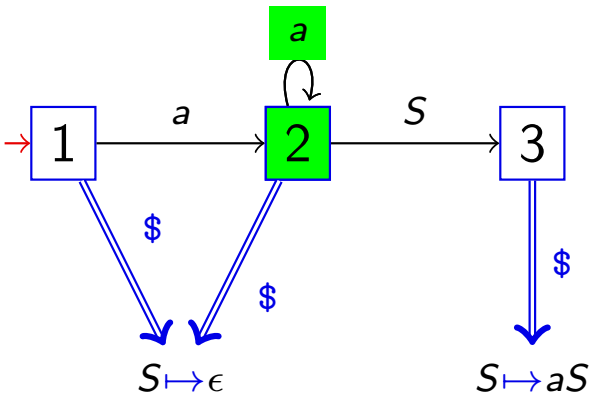
Mot lu : a a a a a a a a a a $

Example of non-linear execution



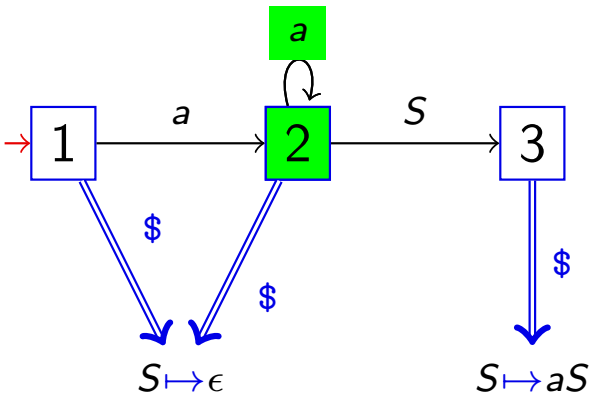
Mot lu : a a a a a a a a a a $

Example of non-linear execution



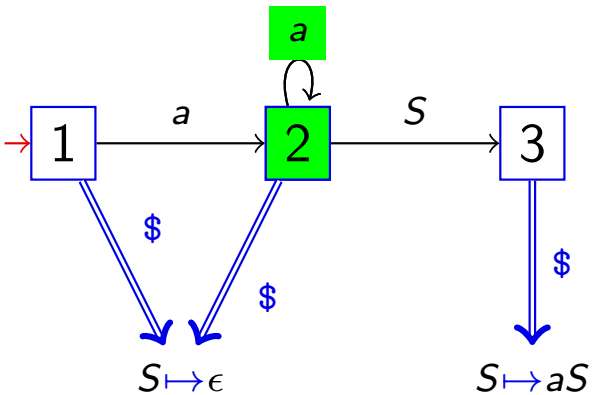
Mot lu : a a a a a a a a a a $

Example of non-linear execution



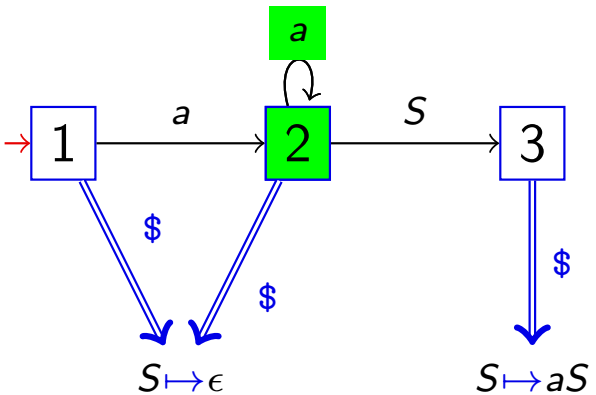
Mot lu : a a a a a a a a a a $

Example of non-linear execution



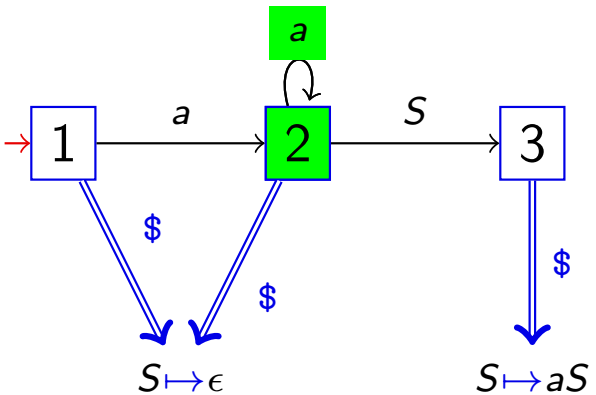
Mot lu : a a a a a a a a a a $

Example of non-linear execution



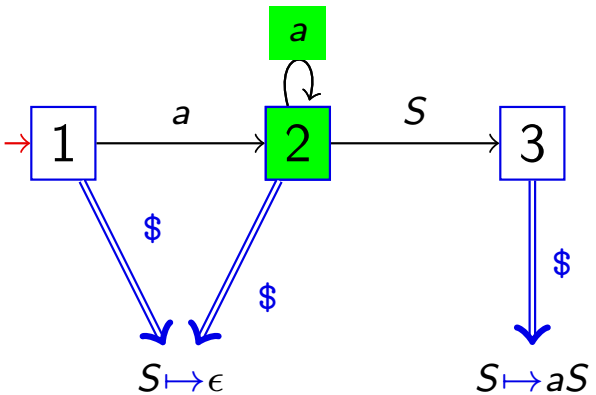
Mot lu : a a a a a a a a a a $

Example of non-linear execution



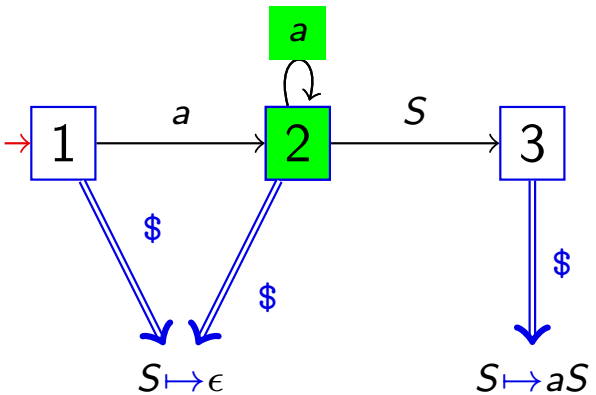
Mot lu : a a a a a a a a a a $

Example of non-linear execution



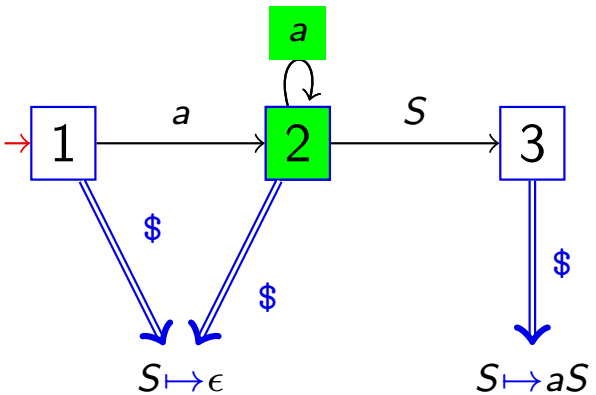
Mot lu : a a a a a a a a a a $

Example of non-linear execution



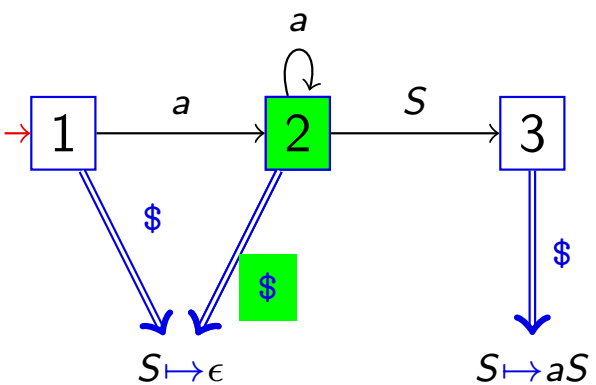
Mot lu : a a a a a a a a a a $

Example of non-linear execution



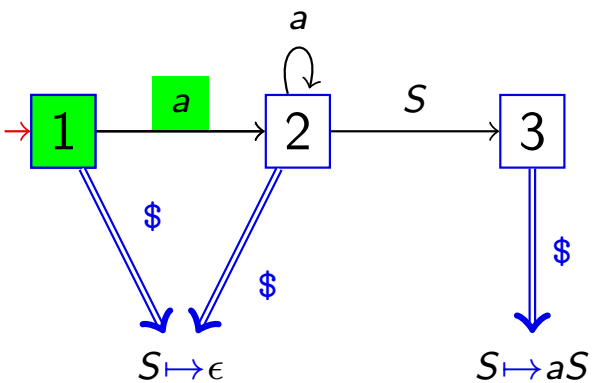
Mot lu : a a a a a a a a a a $

Example of non-linear execution



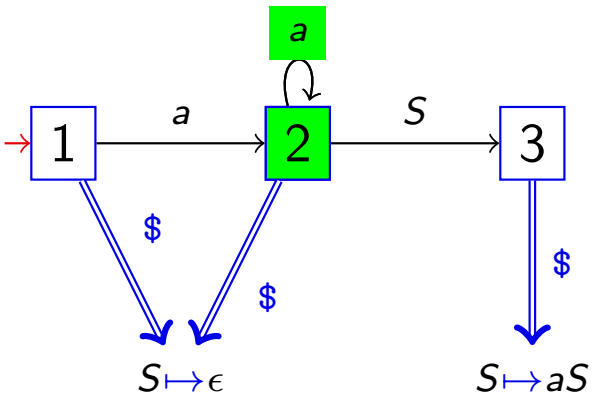
Mot lu : a a a a a a a a a a $

Example of non-linear execution



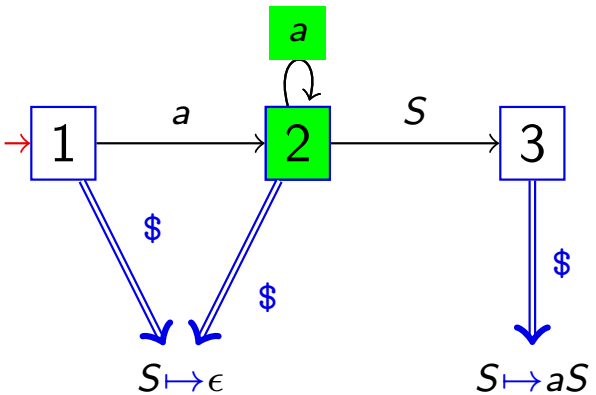
Mot lu : a a a a a a a a a a S \$

Example of non-linear execution



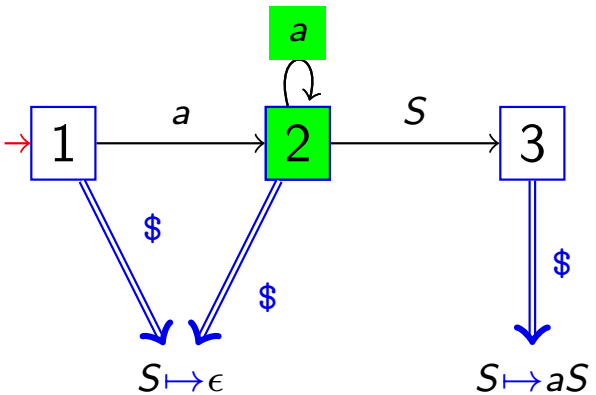
Mot lu : a a a a a a a a a a S \$

Example of non-linear execution



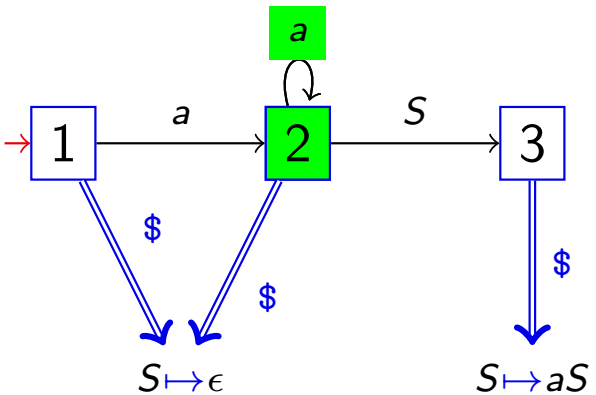
Mot lu : a a a a a a a a a a S \$

Example of non-linear execution



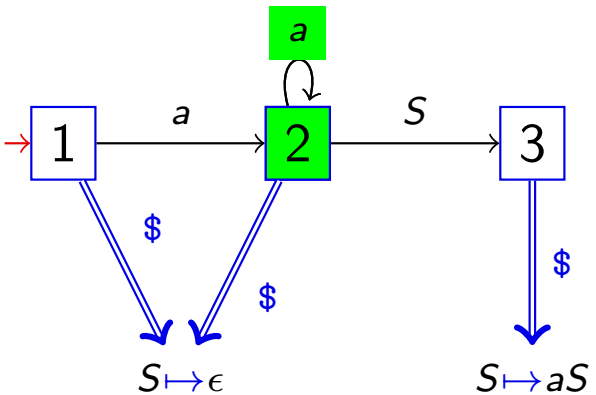
Mot lu : a a a a a a a a a a S \$

Example of non-linear execution



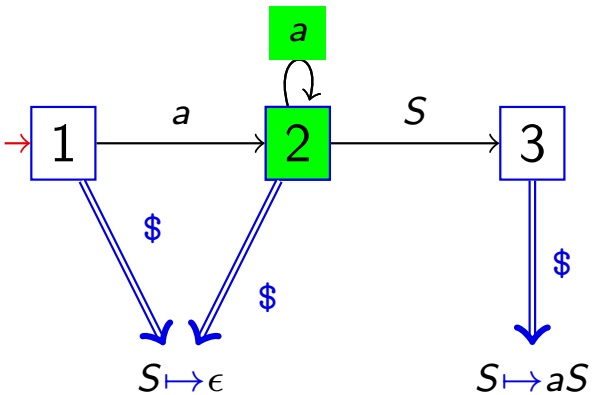
Mot lu : a a a a a a a a a a S \$

Example of non-linear execution



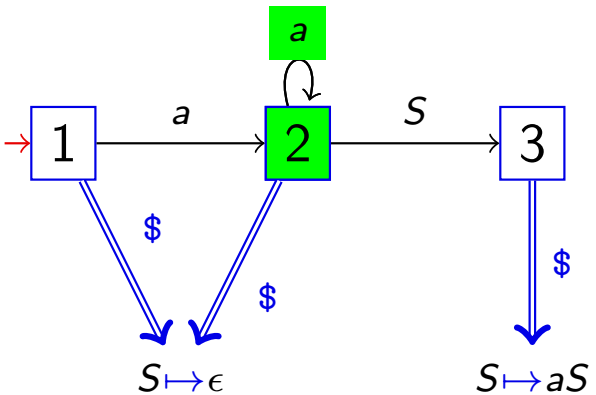
Mot lu : a a a a a a a a a a S $

Example of non-linear execution



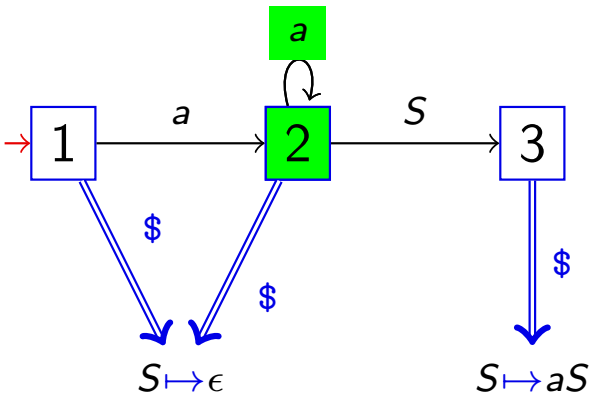
Mot lu : a a a a a a a a a a S $

Example of non-linear execution



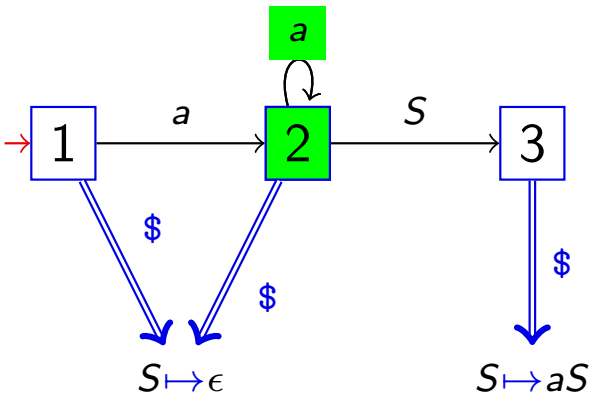
Mot lu : a a a a a a a a a a S \$

Example of non-linear execution



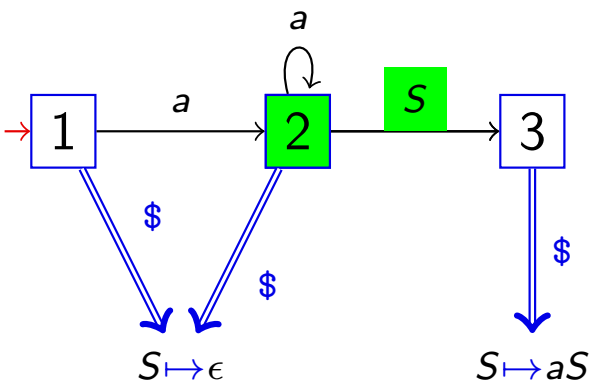
Mot lu : a a a a a a a a a a S \$

Example of non-linear execution



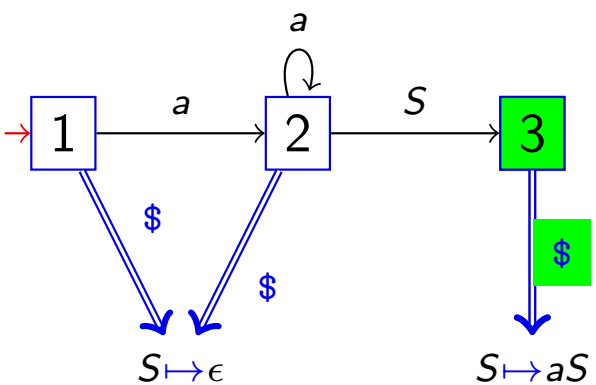
Mot lu : a a a a a a a a a a S $

Example of non-linear execution



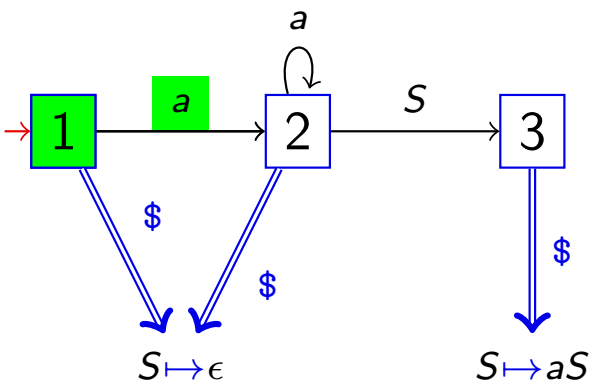
Mot lu : a a a a a a a a a a S $

Example of non-linear execution



Mot lu : a a a a a a a a a a S §

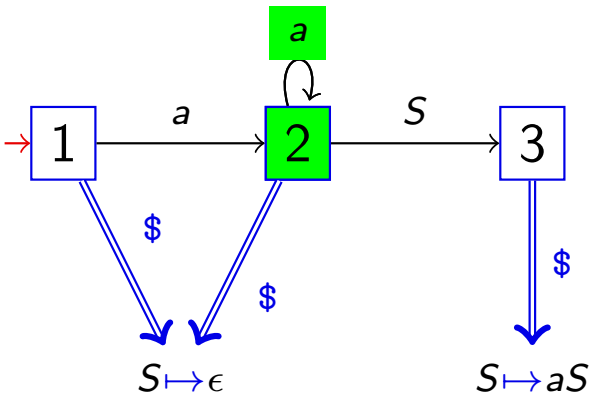
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

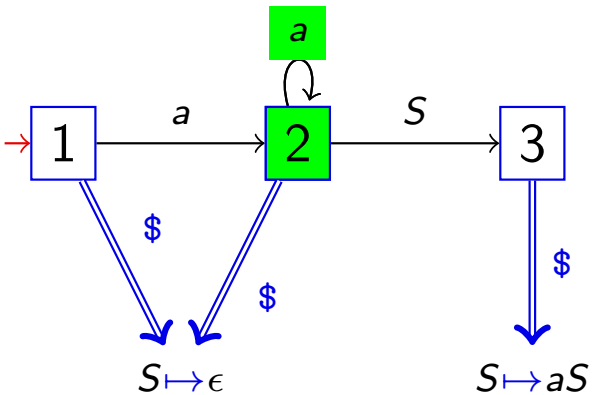
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

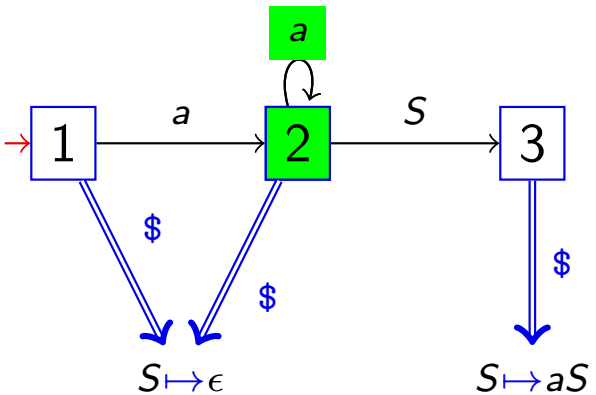
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

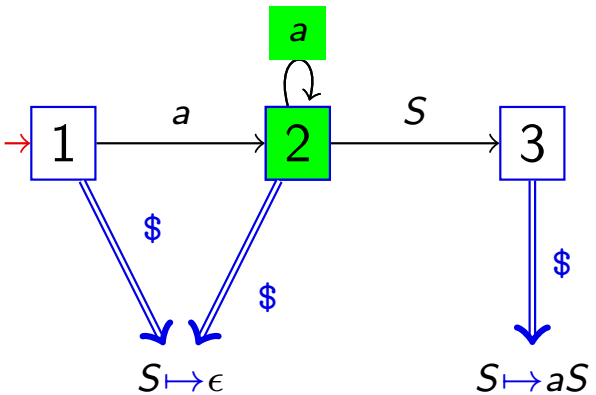
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

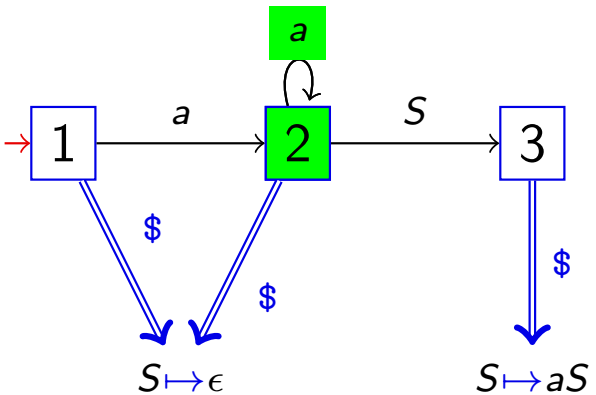
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

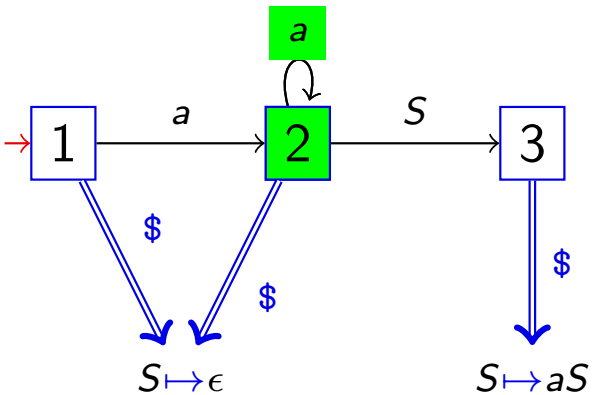
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

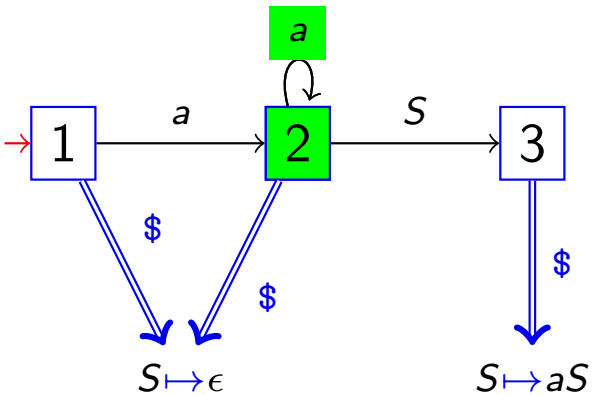
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

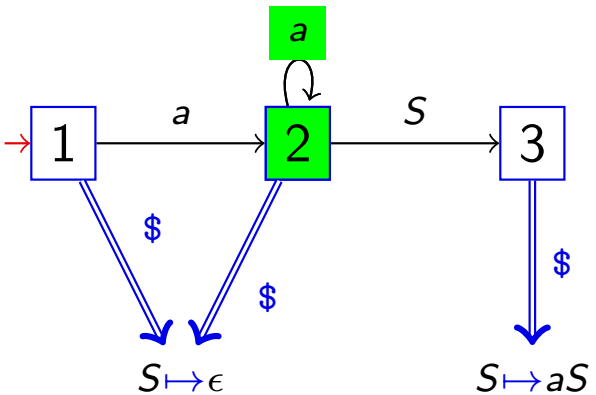
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

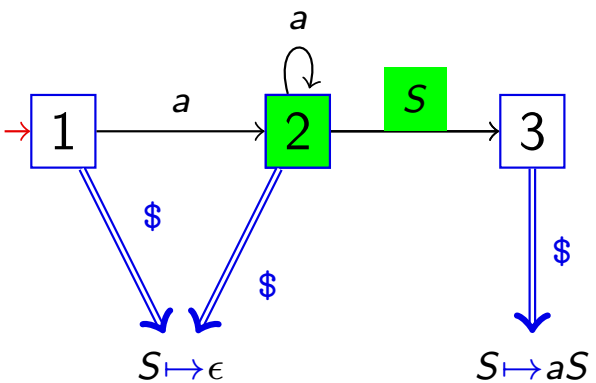
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

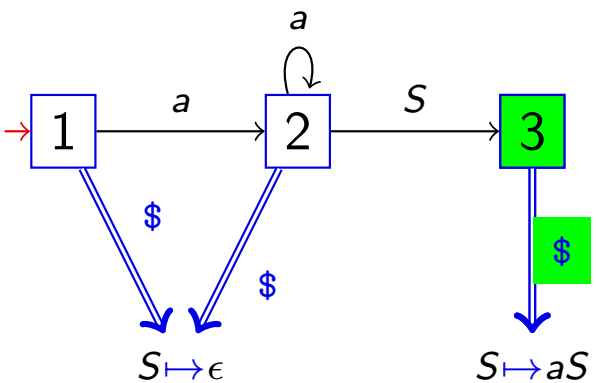
Example of non-linear execution



Mot lu : a a a a a a a a a S $

a S

Example of non-linear execution



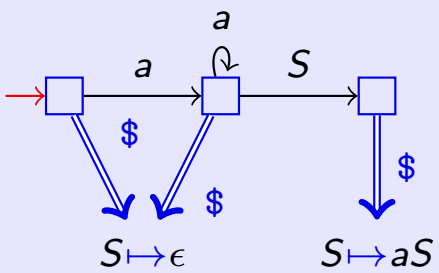
Mot lu : a a a a a a a a a S $

a S

Optimization : linear time

The constructed SLR automaton can be of quadratic complexity

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

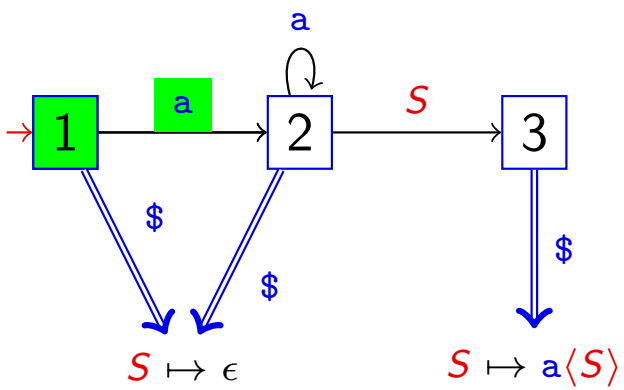


Solution

- In the buffer (which is now a stack), also memorizes the id of the state we came from.
- at the end of reduction, the automaton is restarted from the state given in the stack and on the created NT.

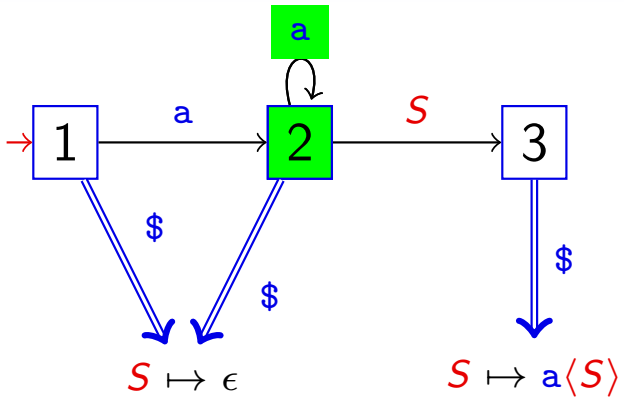
This is a stack automaton (with actions).

Example of linear execution



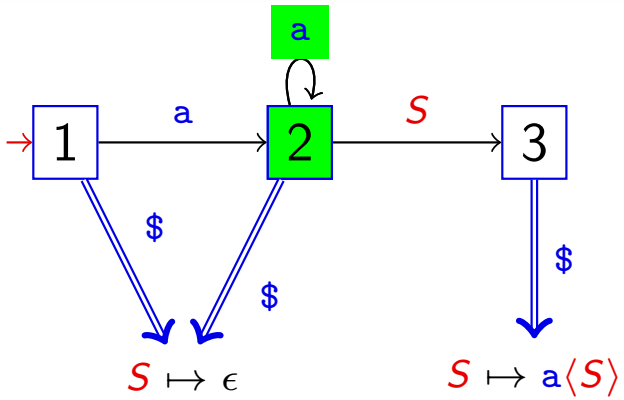
Mot lu : 1 a a a a a a a a a a $

Example of linear execution



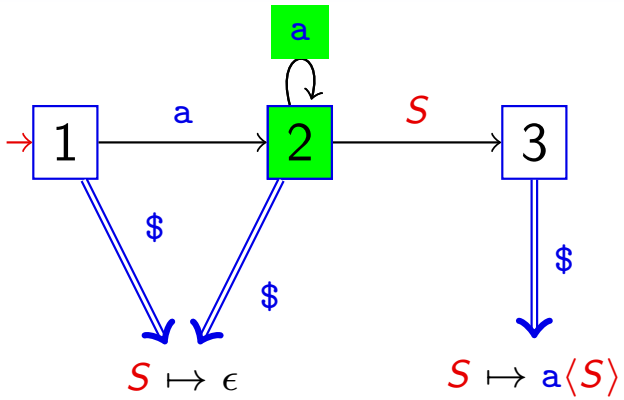
Mot lu : 1 a 2 a a a a a a a a a $

Example of linear execution



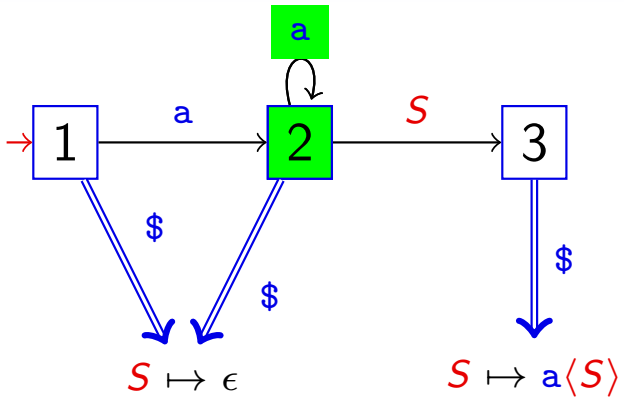
Mot lu : 1 a 2 a 2 a a a a a a a a $

Example of linear execution



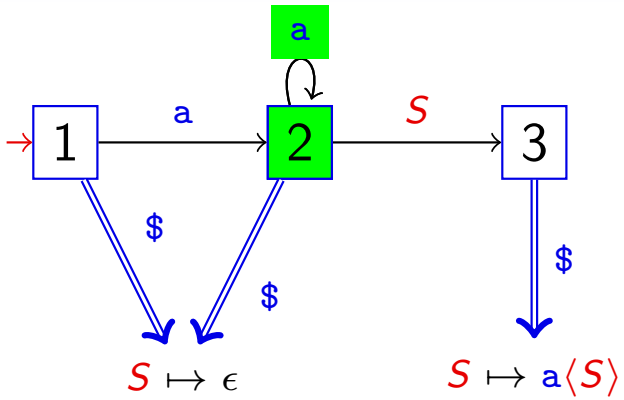
Mot lu : 1 a 2 a 2 a 2 a a a a a a a $

Example of linear execution



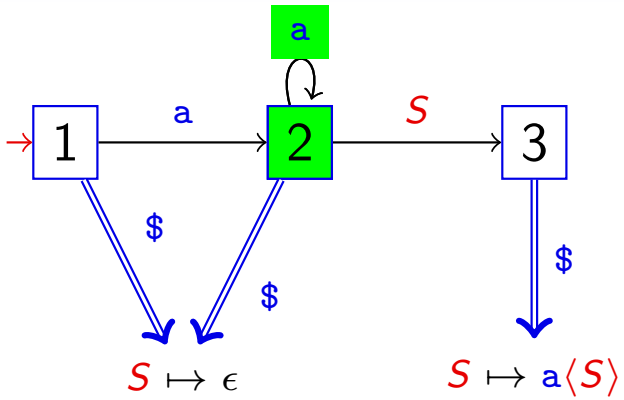
Mot lu : 1 a 2 a 2 a 2 a 2 a a a a a a $

Example of linear execution



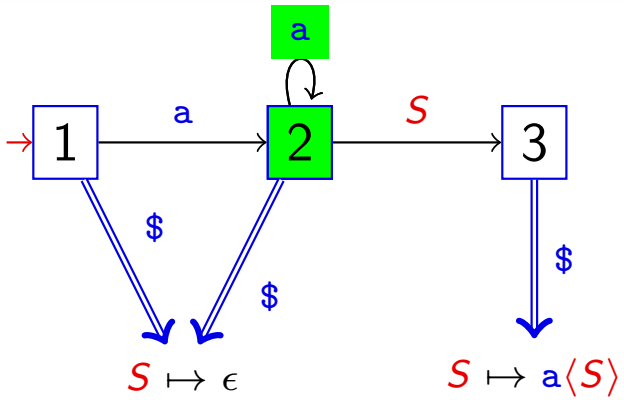
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a a a a a $

Example of linear execution



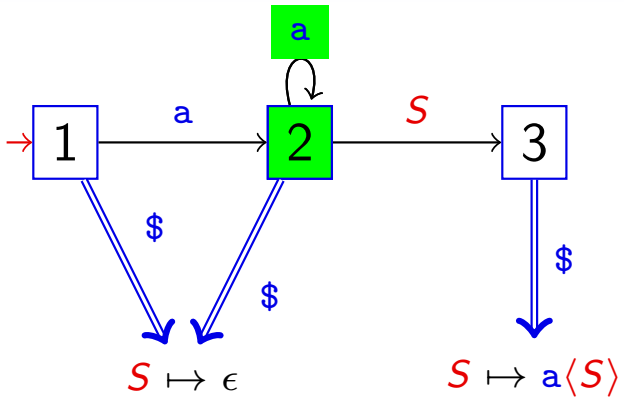
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a a a a $

Example of linear execution



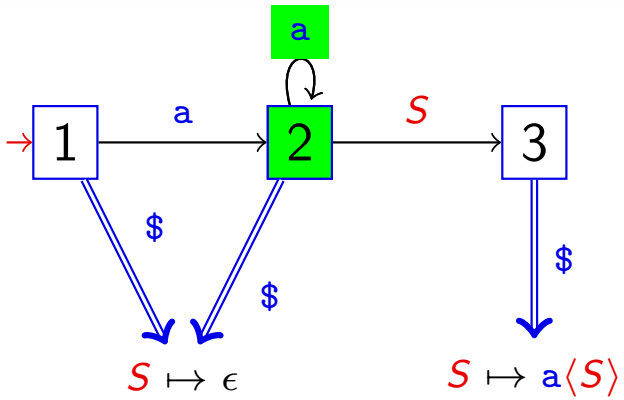
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a a a $

Example of linear execution



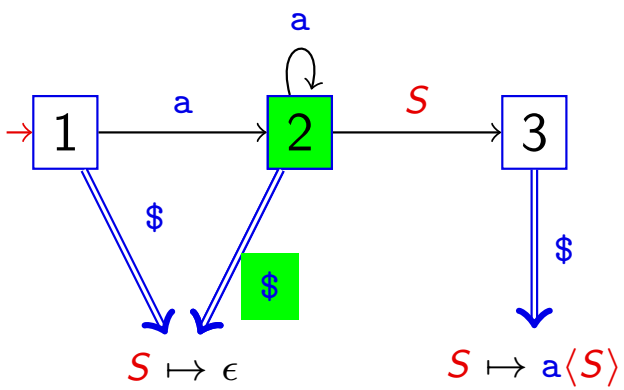
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a a $

Example of linear execution



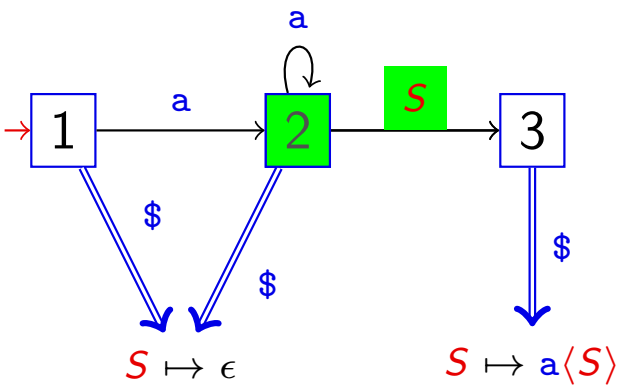
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a $

Example of linear execution



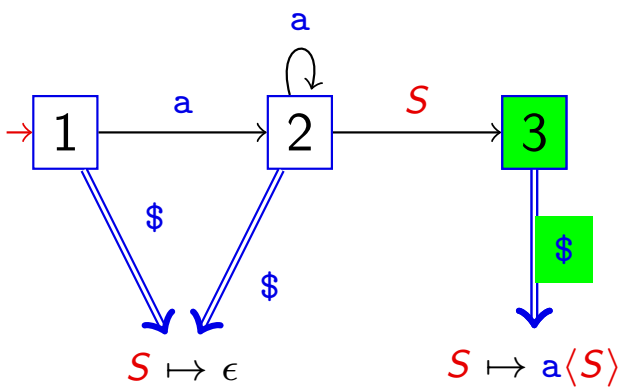
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 $

Example of linear execution



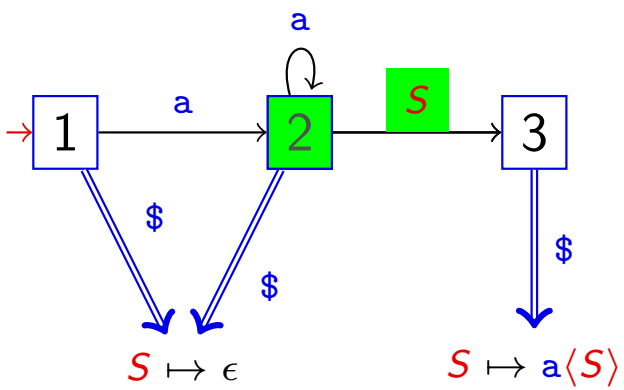
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S $

Example of linear execution



Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S 3 $

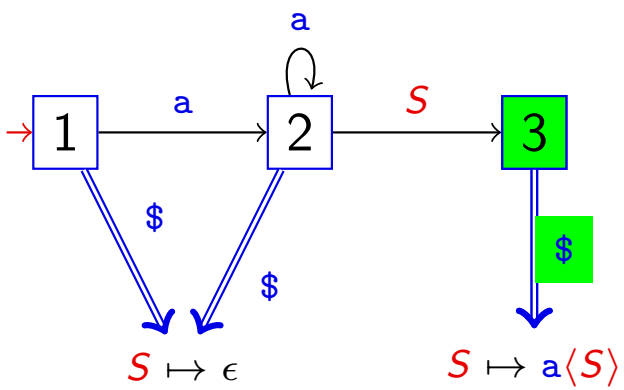
Example of linear execution



Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S $

a S

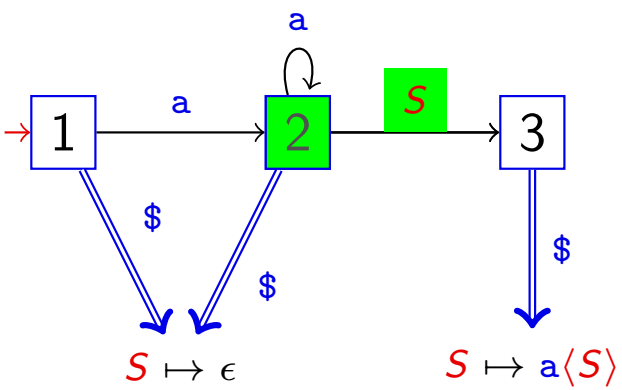
Example of linear execution



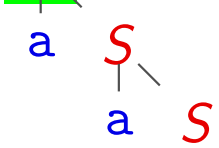
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S 3 $

a S

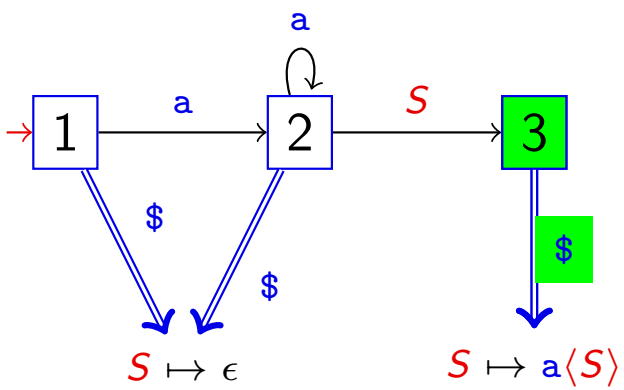
Example of linear execution



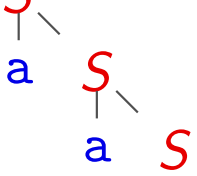
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S $



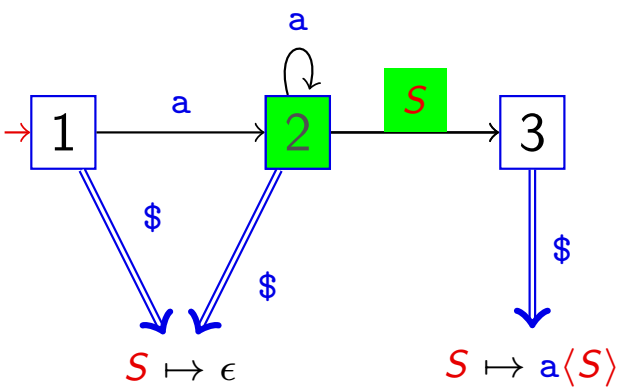
Example of linear execution



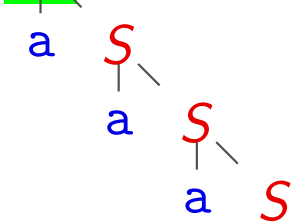
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S 3 $



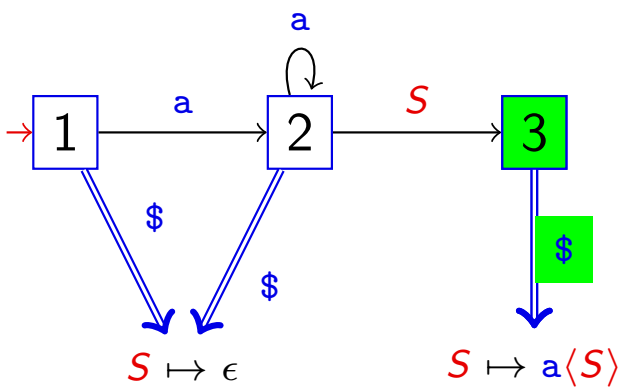
Example of linear execution



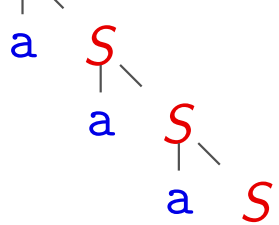
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S $



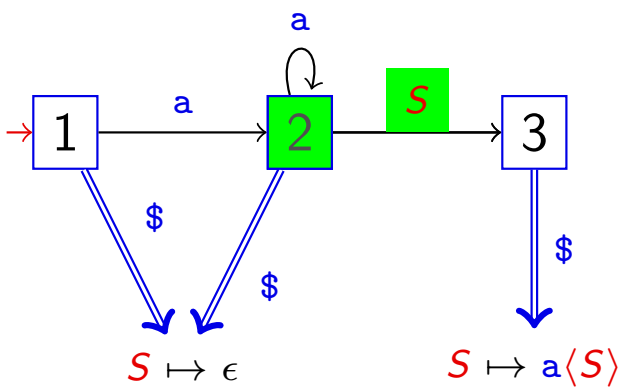
Example of linear execution



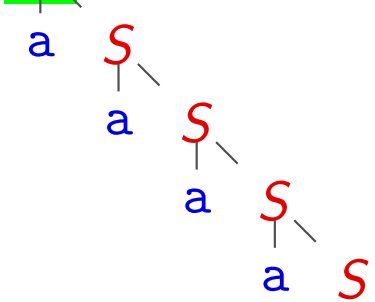
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 a 2 S 3 $



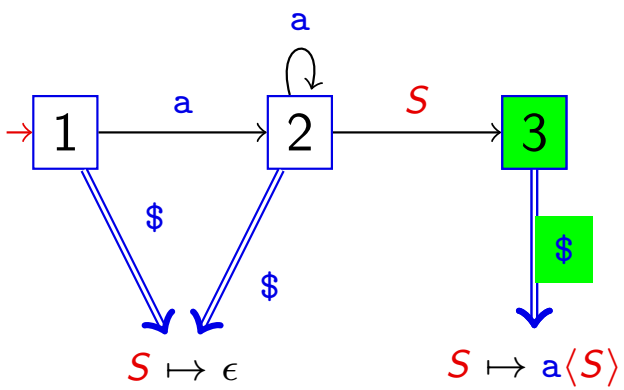
Example of linear execution



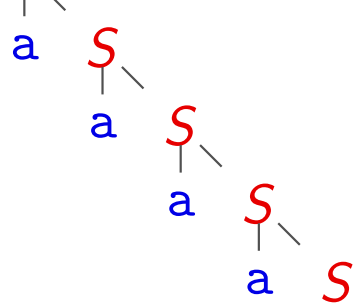
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 S $



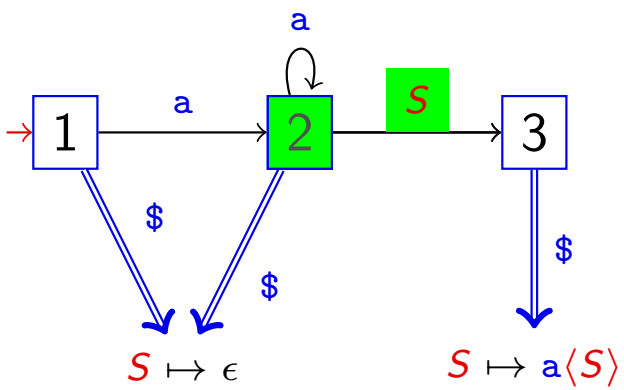
Example of linear execution



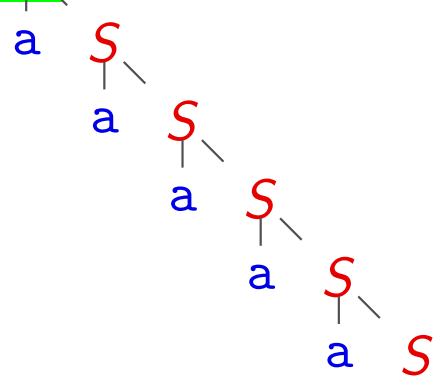
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 a 2 S 3 $



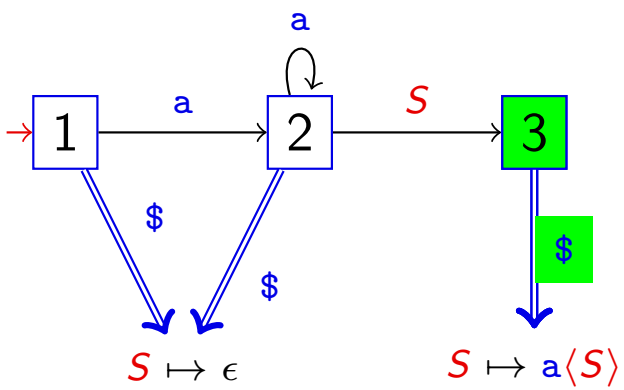
Example of linear execution



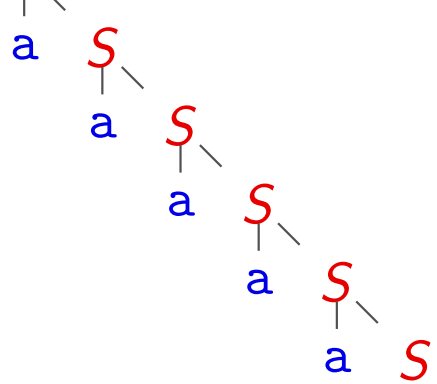
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 S $



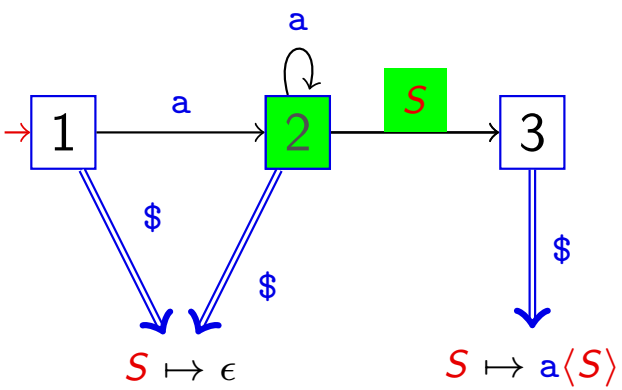
Example of linear execution



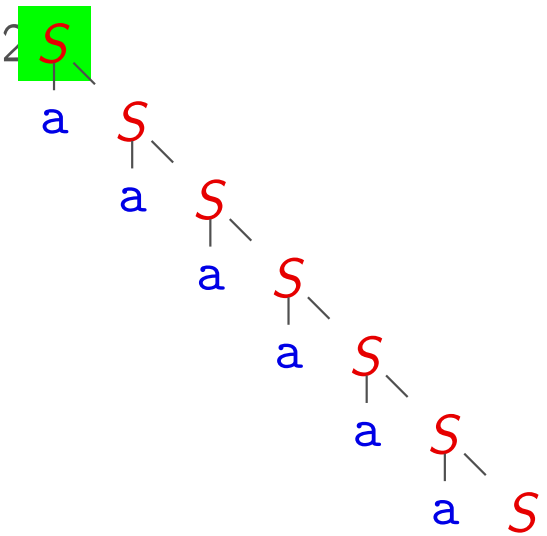
Mot lu : 1 a 2 a 2 a 2 a 2 a 2 S 3 $



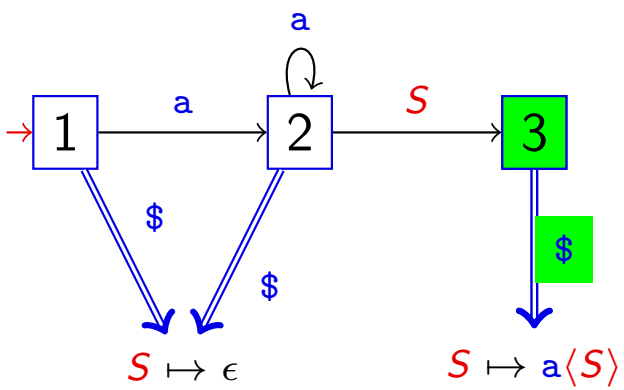
Example of linear execution



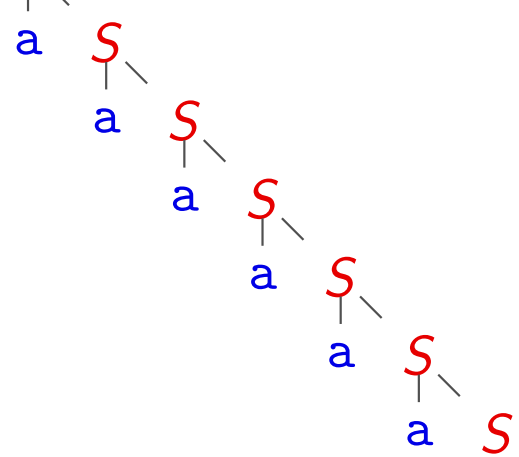
Mot lu : 1 a 2 a 2 a 2 a 2 S $



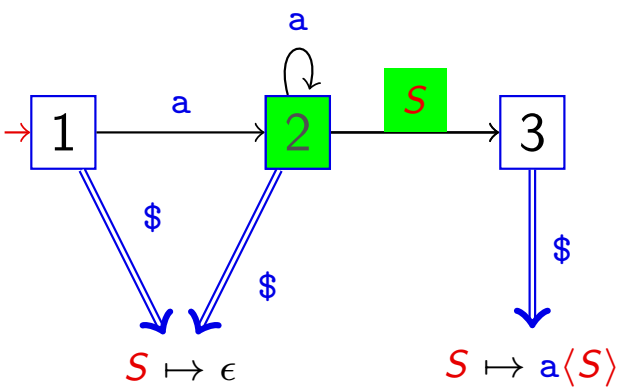
Example of linear execution



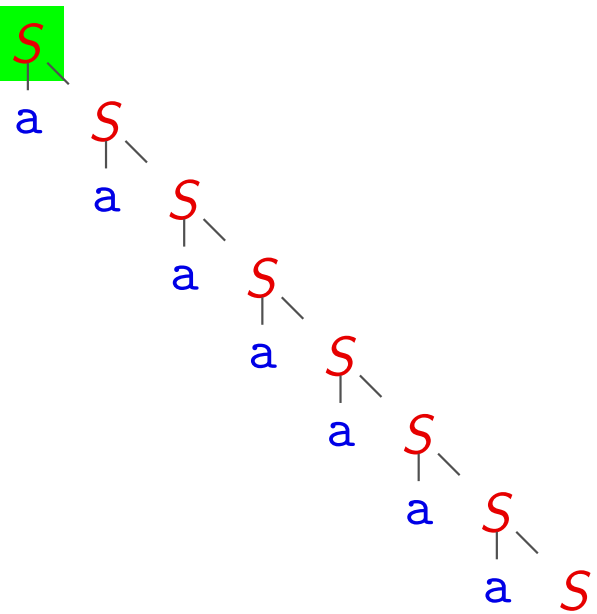
Mot lu : 1 a 2 a 2 a 2 a 2 S 3 $



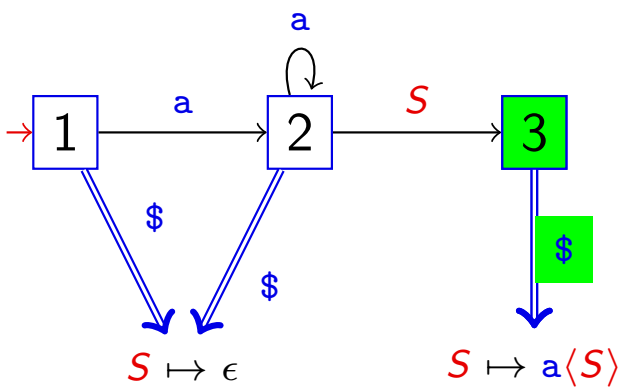
Example of linear execution



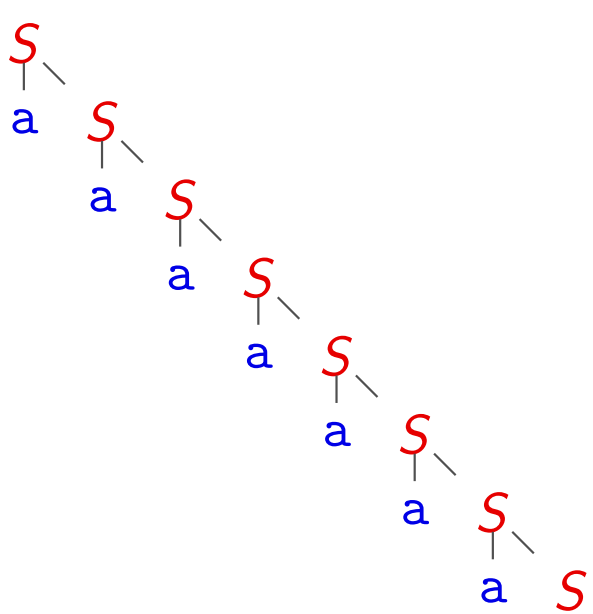
Mot lu : 1 a 2 a 2 a 2 S $



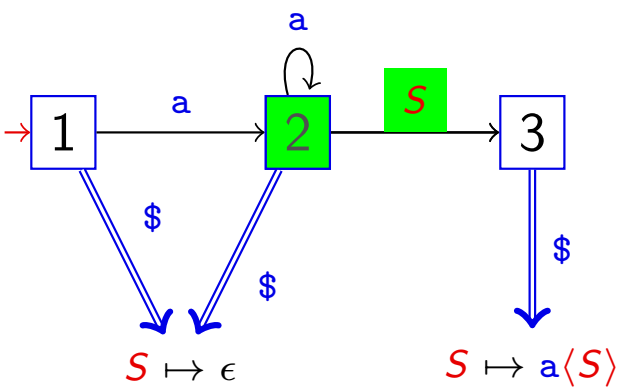
Example of linear execution



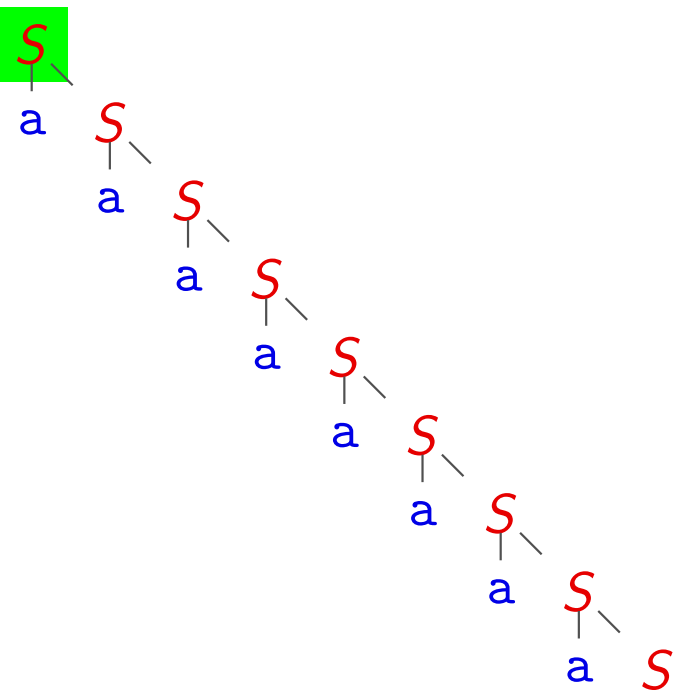
Mot lu : 1 a 2 a 2 a 2 S 3 $



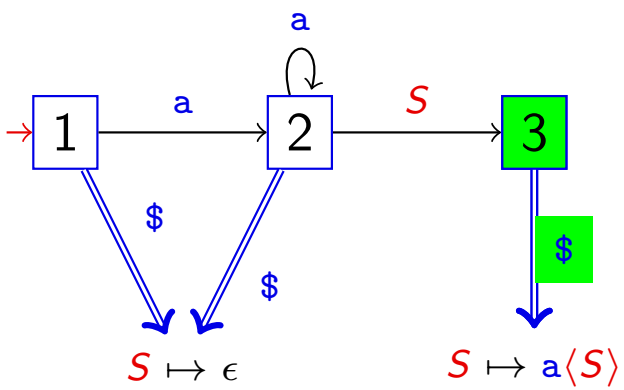
Example of linear execution



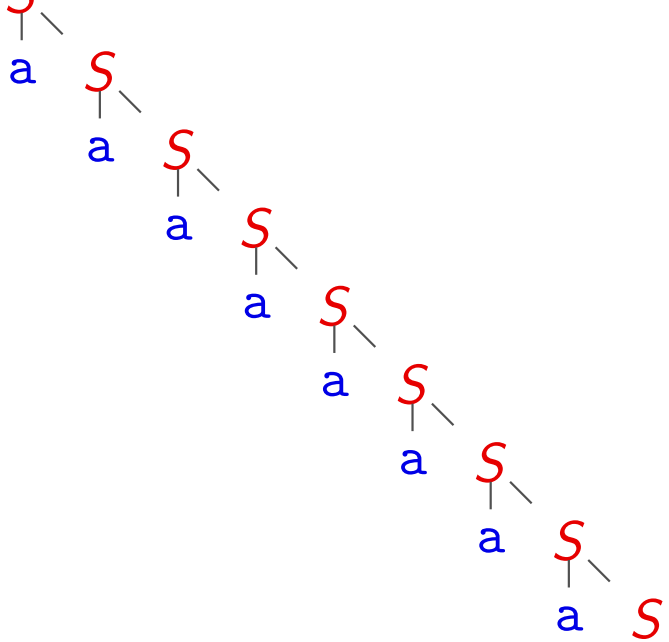
Mot lu : 1 a 2 a 2 S $



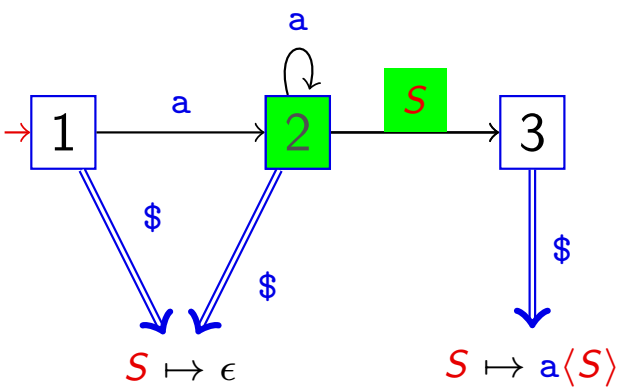
Example of linear execution



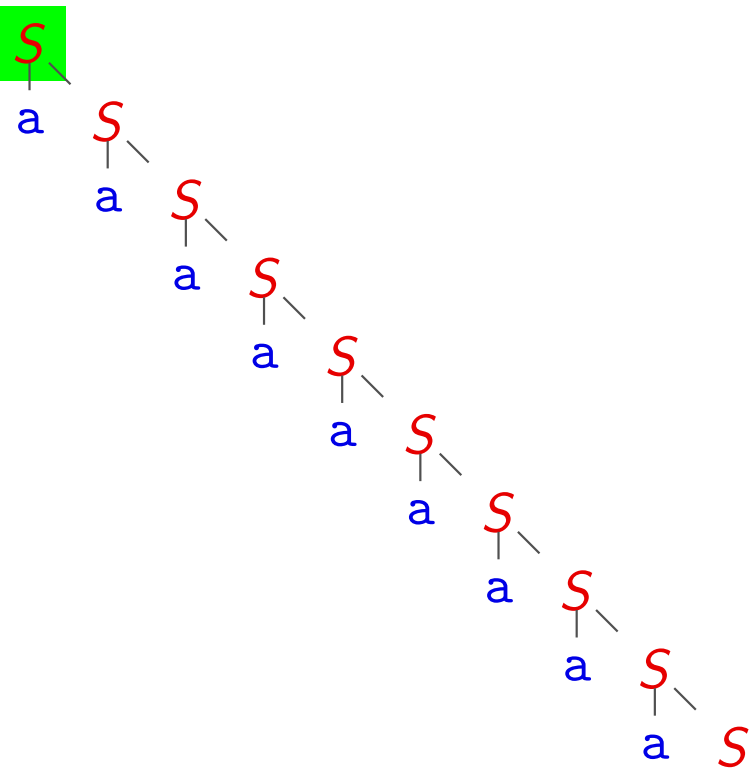
Mot lu : 1 a 2 a 2 S 3 $



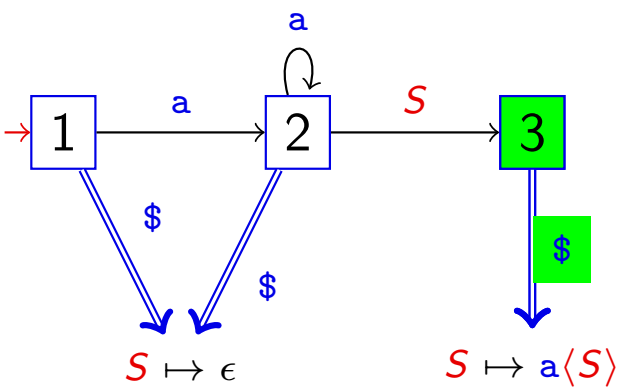
Example of linear execution



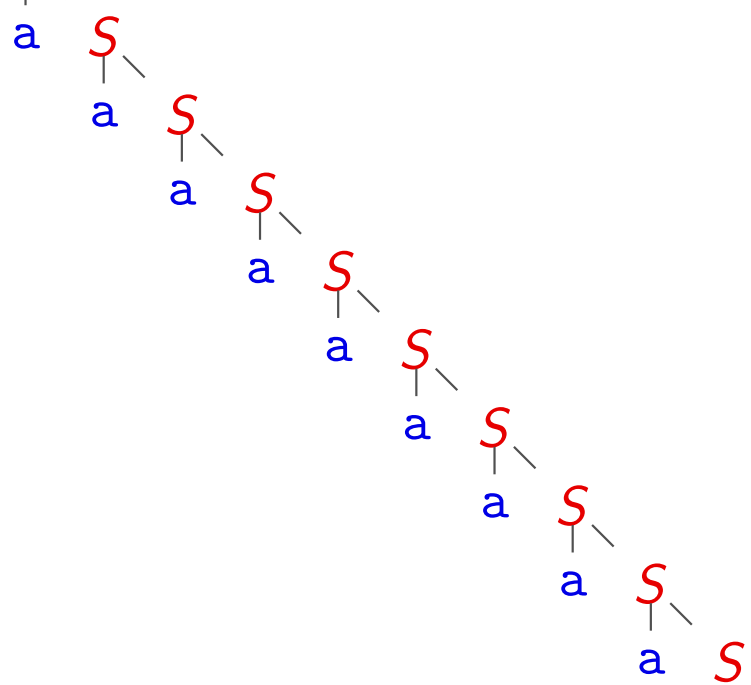
Mot lu : 1 a 2 S $



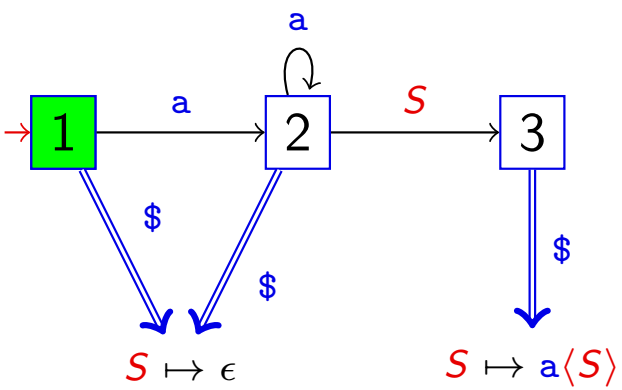
Example of linear execution



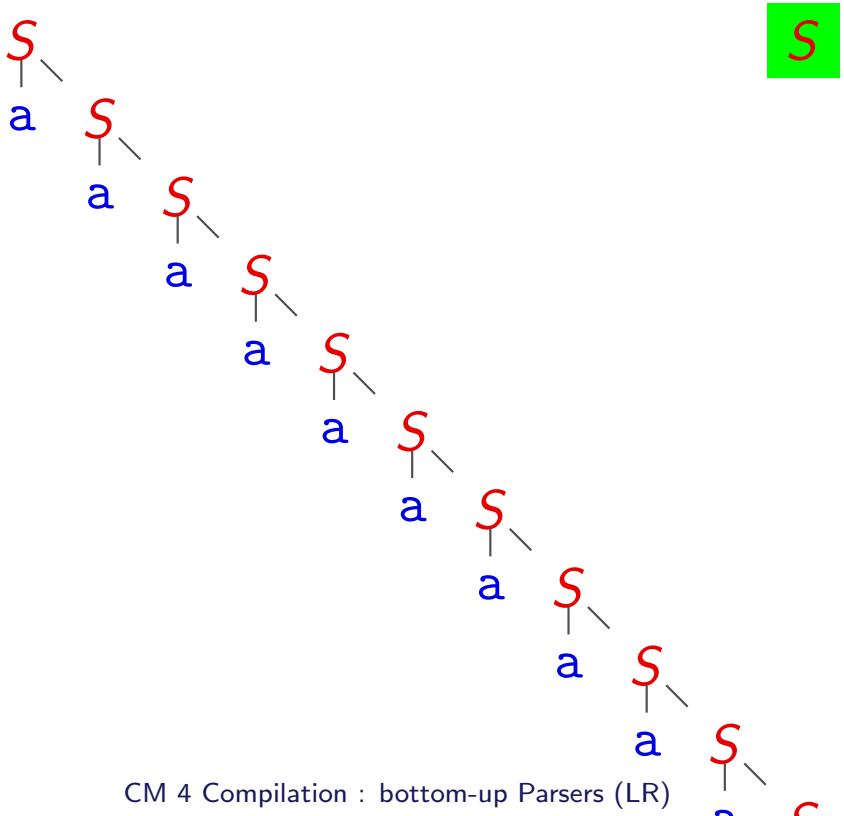
Mot lu : 1 a 2 S 3 $



Example of linear execution

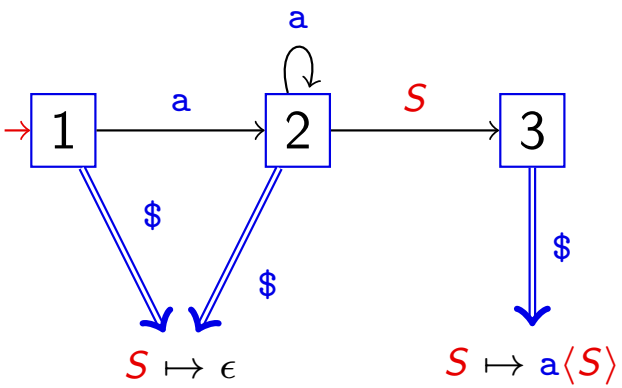


Mot lu :

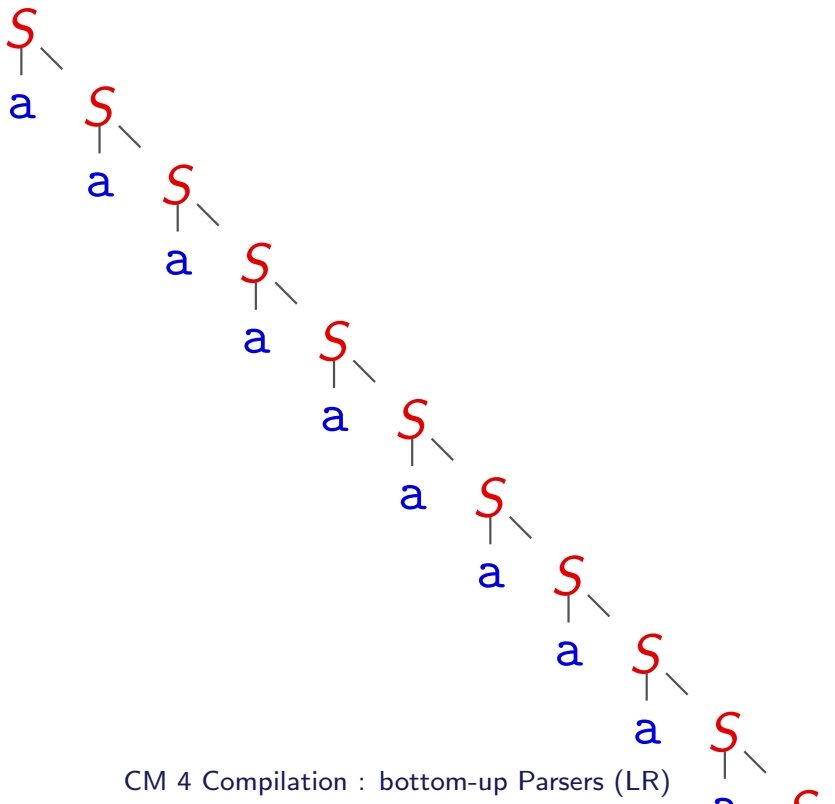


S $

Example of linear execution



Mot lu : $



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".