CSL302: Compiler Design

Bottom Up Parsing

Vishwesh Jatala

Assistant Professor

Department of CSE

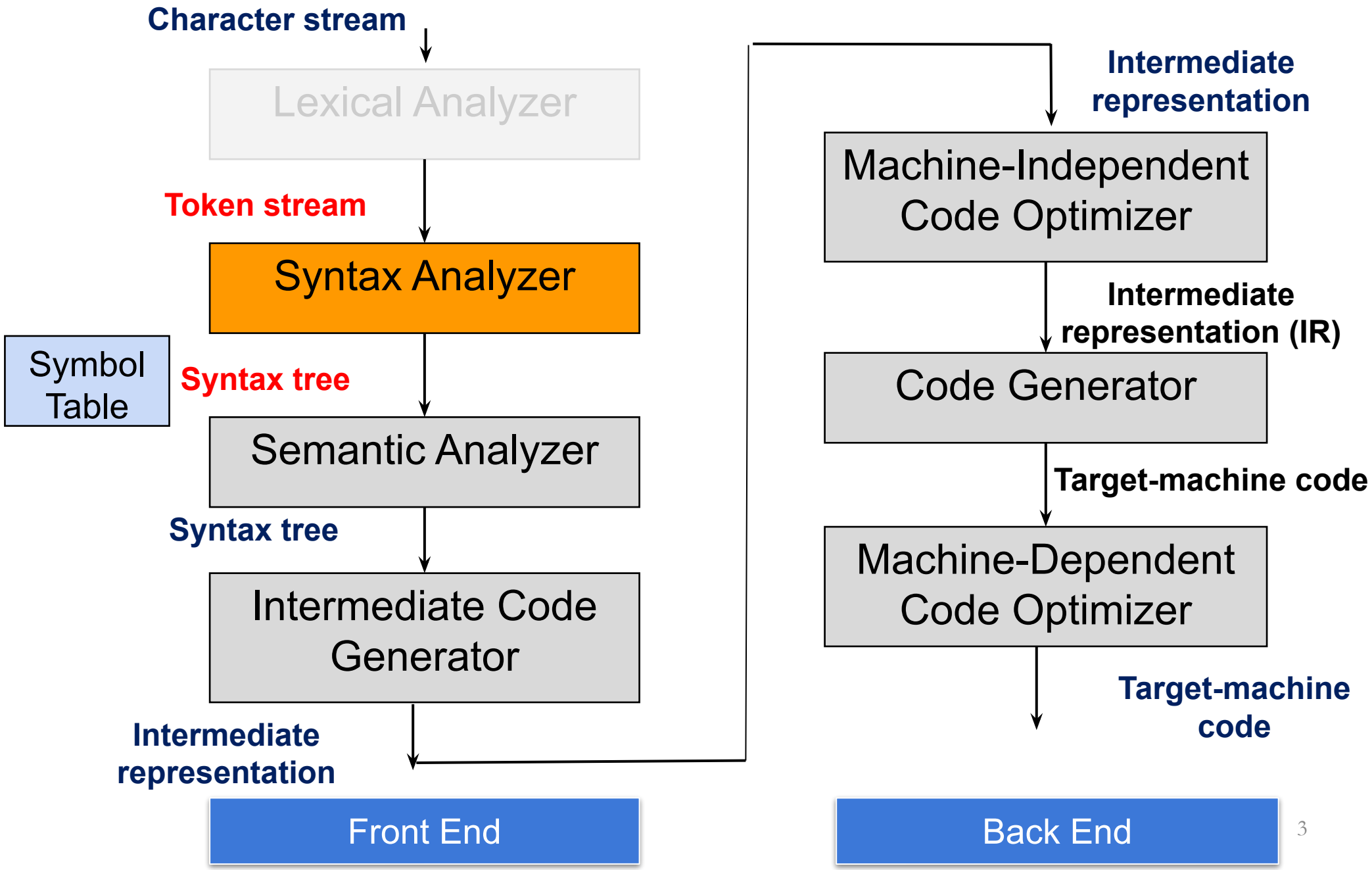
Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

Acknowledgement

- Today's slides are modified from that of *Stanford University:*
  - *https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/*

Compiler Design



Analysis of SLR(1)

- Exploits lookahead in a small space.
  - Small automaton – same number of states as in as LR(0).
  - Works on many more grammars than LR(0)
- Too weak for most grammars: lose context from not having extra states.

Example

id	=	*	id

S → E

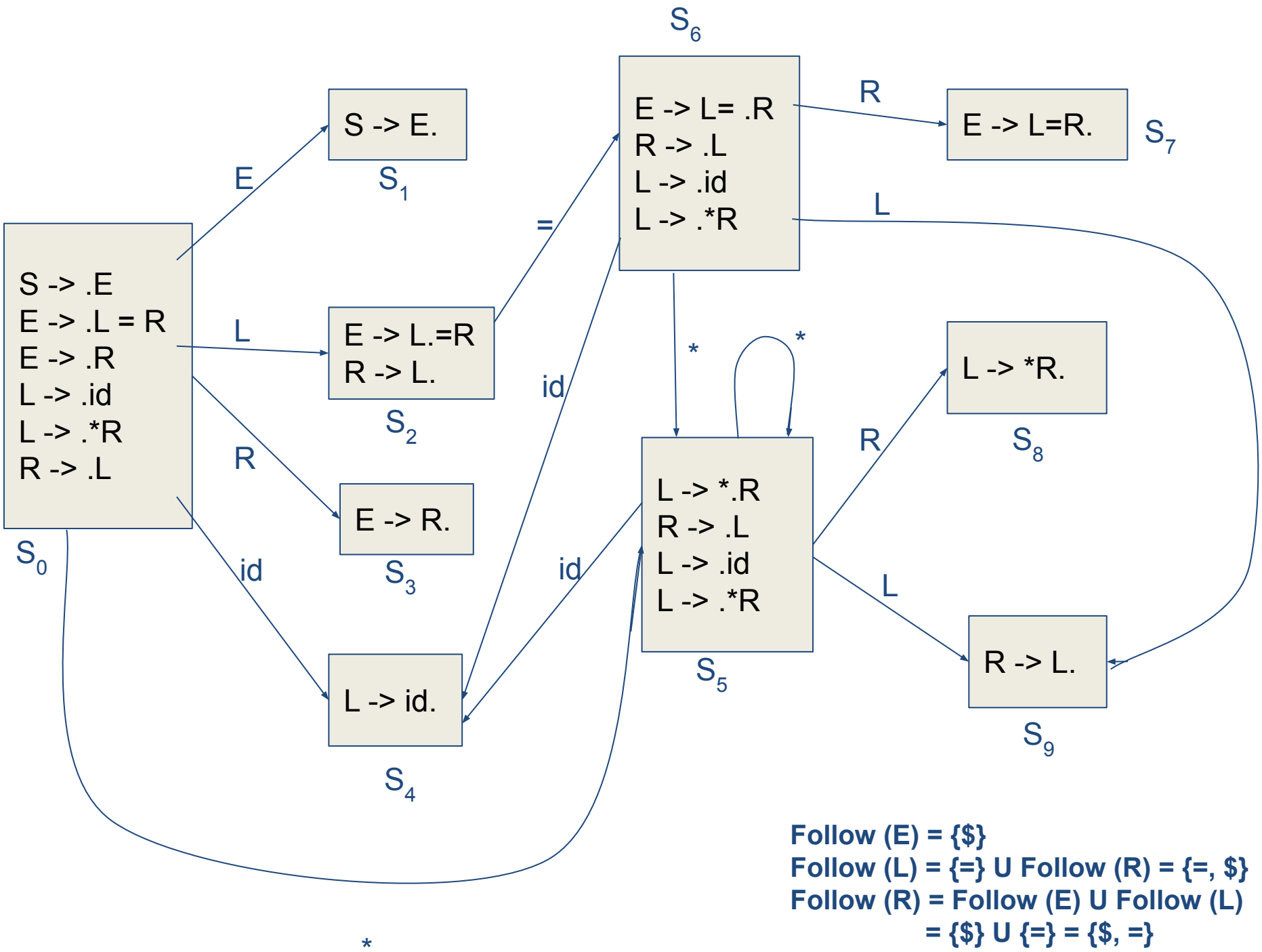
E → L = R

E → R

L → id

L → *R

R → L


S -> .E
E -> .L = R
E -> .R
L -> .id
L -> .*R
R -> .L
S0
E
L
R
id
S -> E.
S1
E -> L.=R
R -> L.
S2
E -> R.
S3
L -> id.
S4
=
E -> L= .R
R -> .L
L -> .id
L -> .*R
S6
R
E -> L=R.
S7
L
id
*
*
L -> *.R
R -> .L
L -> .id
L -> .*R
S5
R
L -> *R.
S8
id
L
R -> L.
S9
*
Follow (E) = {$}
Follow (L) = {=} U Follow (R) = {=, $}
Follow (R) = Follow (E) U Follow (L)
= {$} U {=} = {$, =}

S → E (1)
E → L = R (2)
E → R (3)
L → id (4)
L → *R (5)
R → L (6)

Follow (S) = {$}
Follow (E) = {$}
Follow (L) = {=, $}
Follow (R) = {=, $}

	id	=	*	$	E	L	R
s_0	s_4		s_5		s_1	s_2	s_3
s_1				r_2			
s_2		s_6 / r_6		r_6			
s_3				r_3			
s_4		r_4		r_4			
s_5	s_4		s_5			s_9	s_8
s_6	s_4		s_5			s_9	s_7
s_7				r_2			
s_8		r_5		r_5			
s_9		r_6		r_6			

The Limits of SLR(1)

$S \rightarrow E$

$E \rightarrow L = R$

$E \rightarrow R$

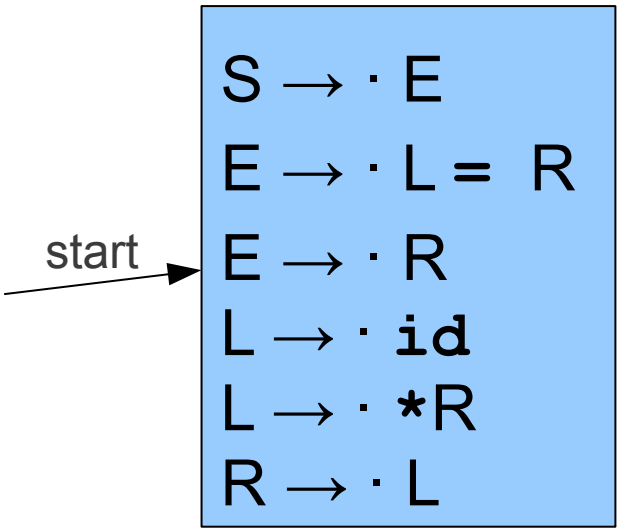
$L \rightarrow id$

$L \rightarrow *R$

$R \rightarrow L$

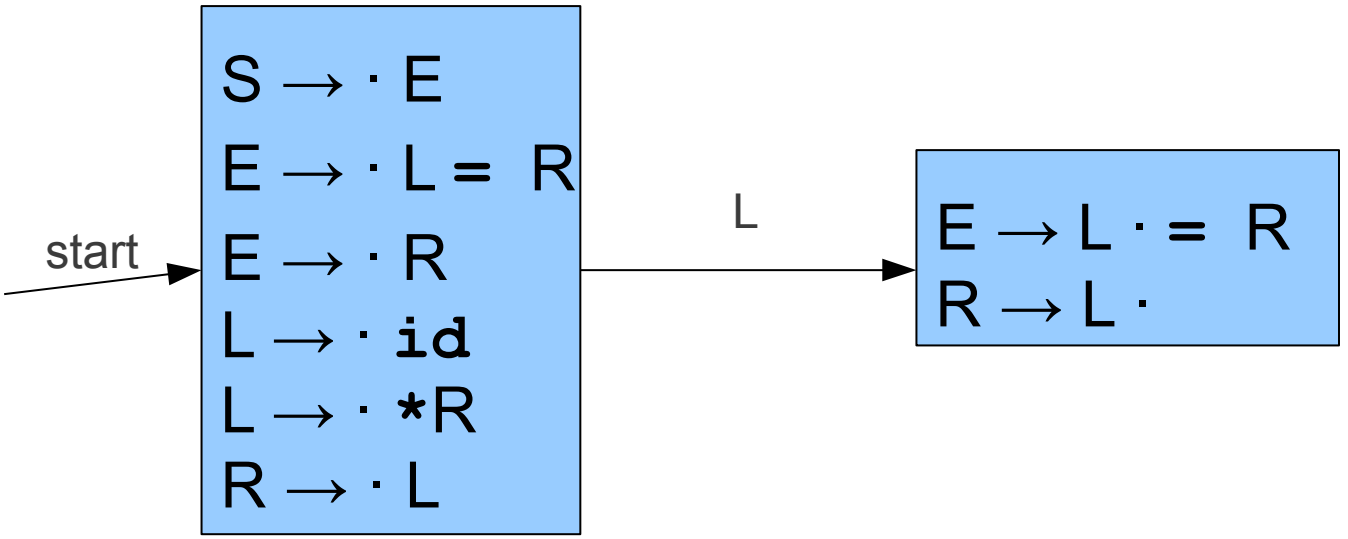
The Limits of SLR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L



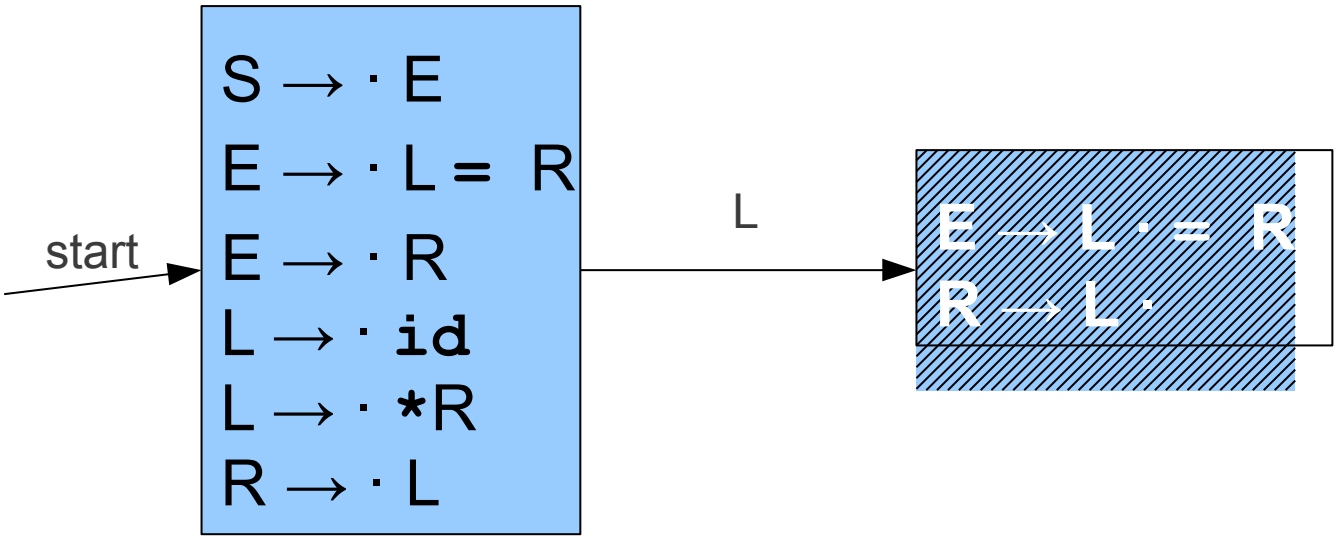
The Limits of SLR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L



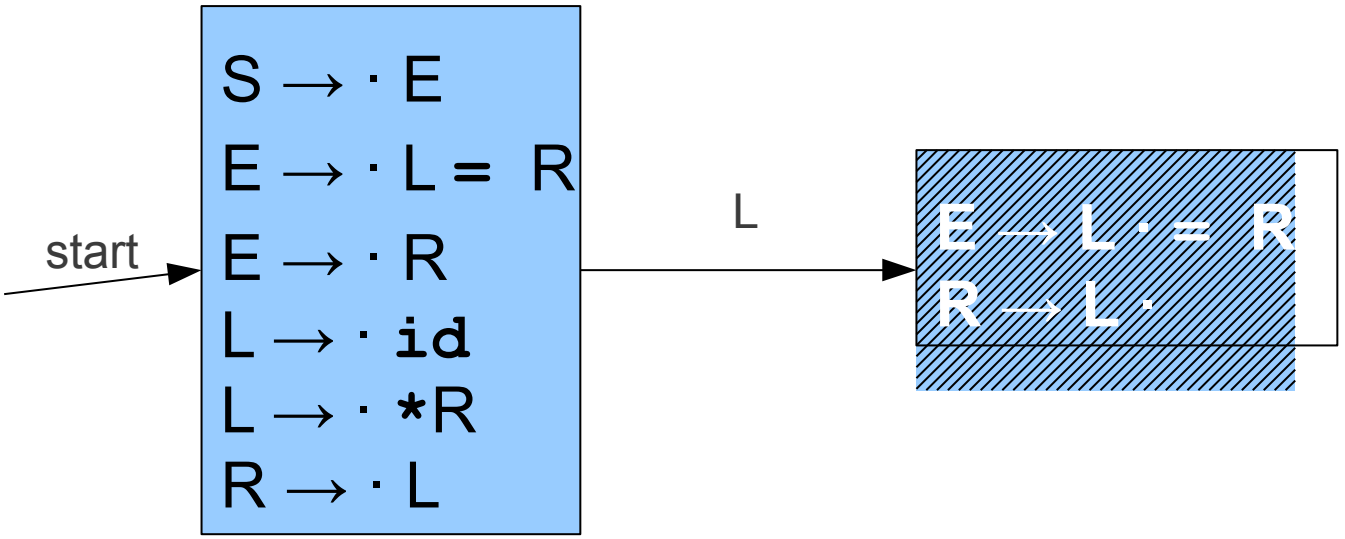
The Limits of SLR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L



The Limits of SLR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L

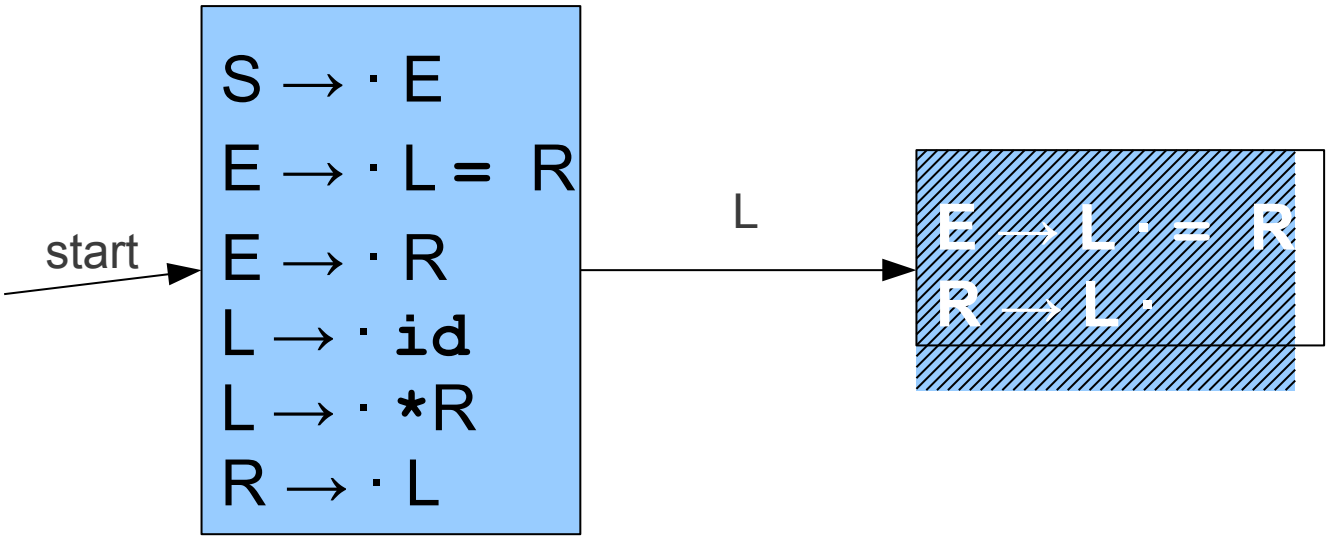


E → L · = R tells us to shift on seeing **=**

R → L · tells us to reduce on FOLLOW(**R**).

The Limits of SLR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L

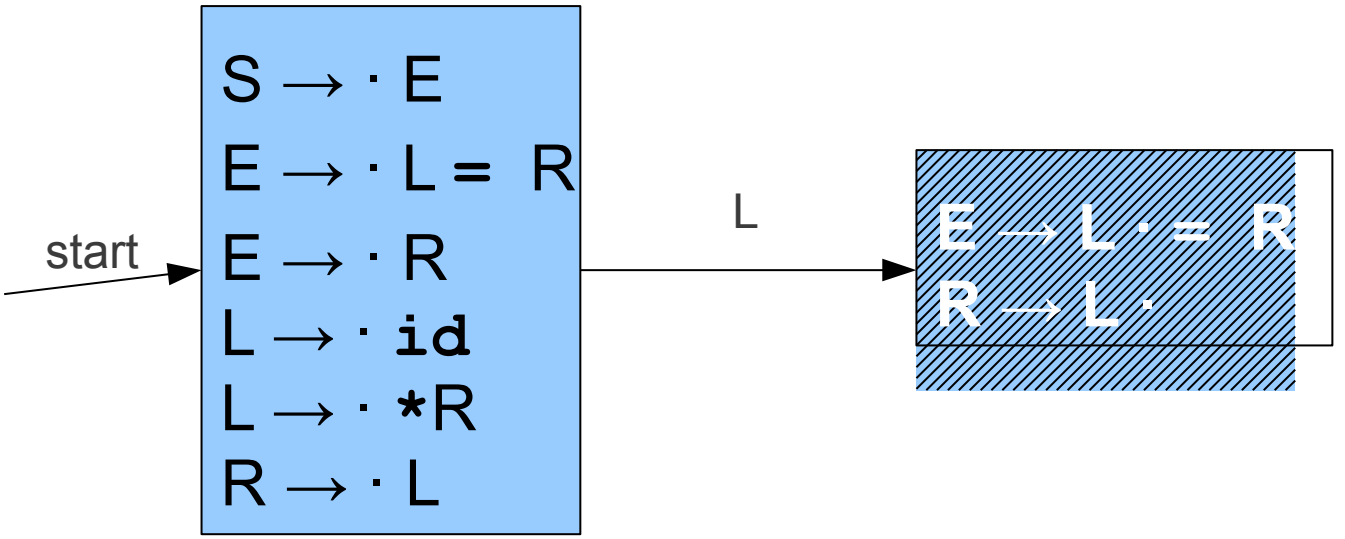


E → L · = R tells us to shift on seeing **=**

R → L · tells us to reduce on FOLLOW(**R**).

The Limits of SLR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L



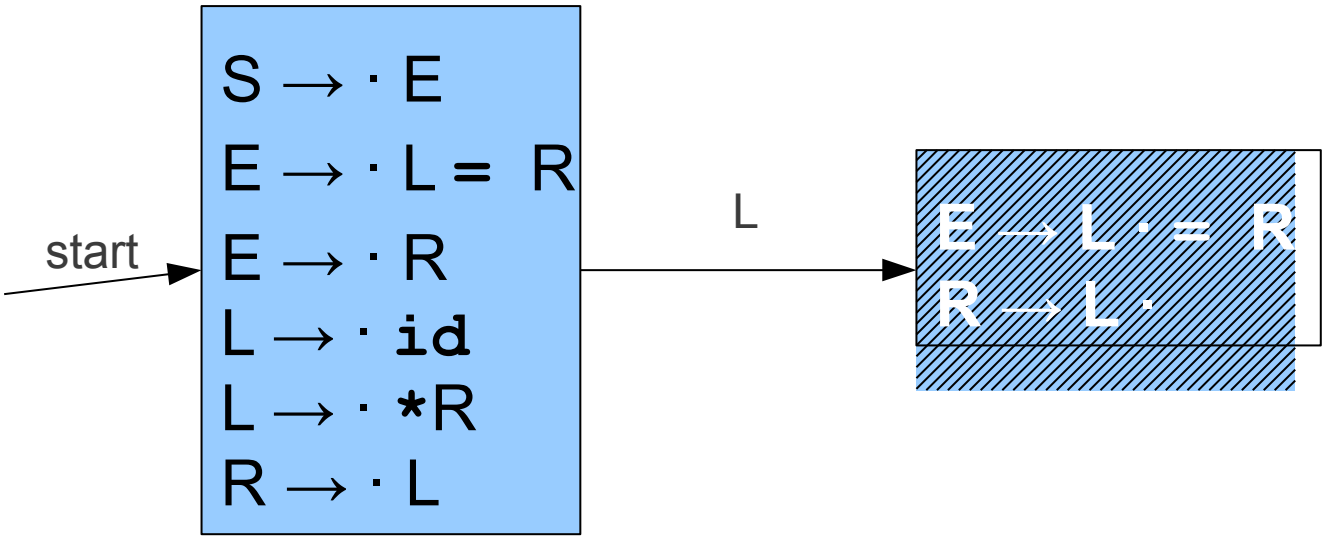
E → L · = R tells us to shift on seeing **=**

R → L · tells us to reduce on FOLLOW(**R**).

= ∈ FOLLOW(**R**).

The Limits of SLR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L



E → L · = R tells us to shift on seeing **=**

R → L · tells us to reduce on FOLLOW(**R**).

= ∈ FOLLOW(**R**).

We have a conflict!

A Lack of Context

S → E

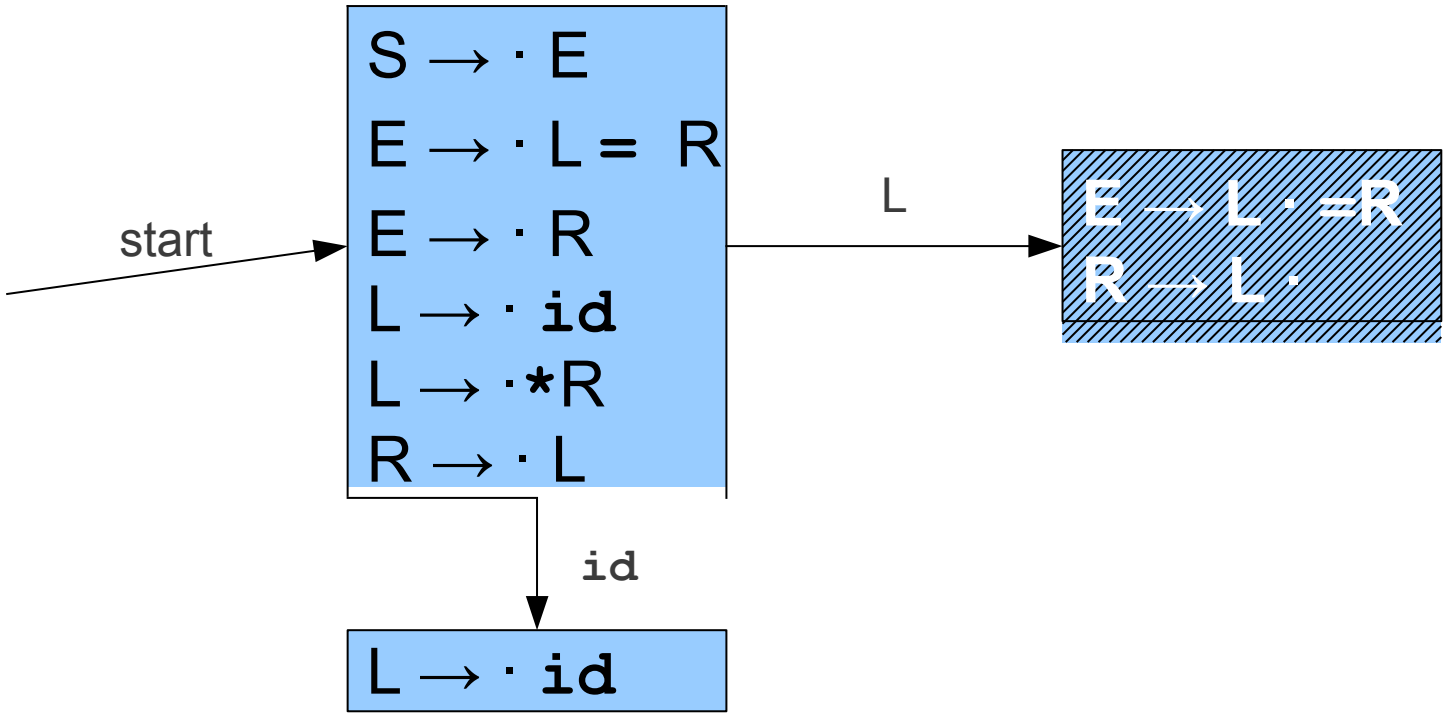
E → L = R

E → R

L → id

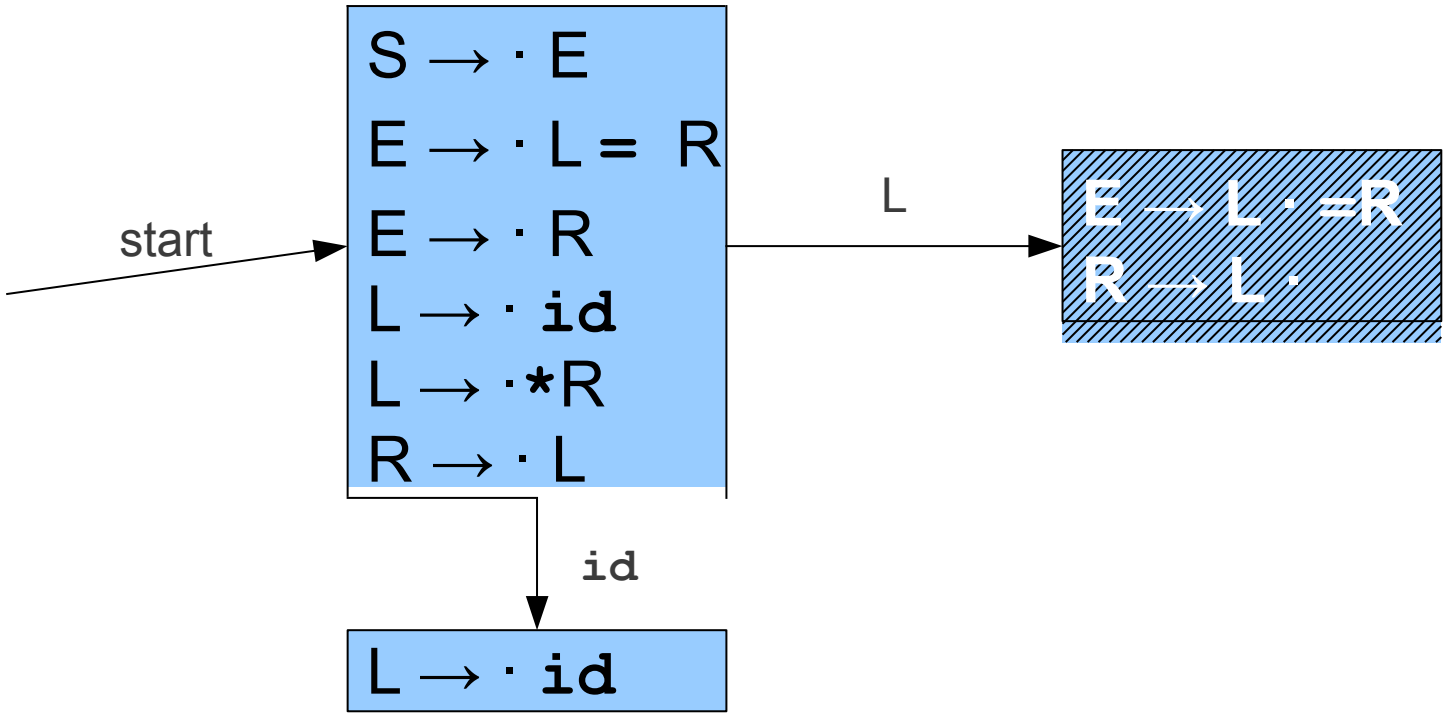
L → *R

R → L



A Lack of Context

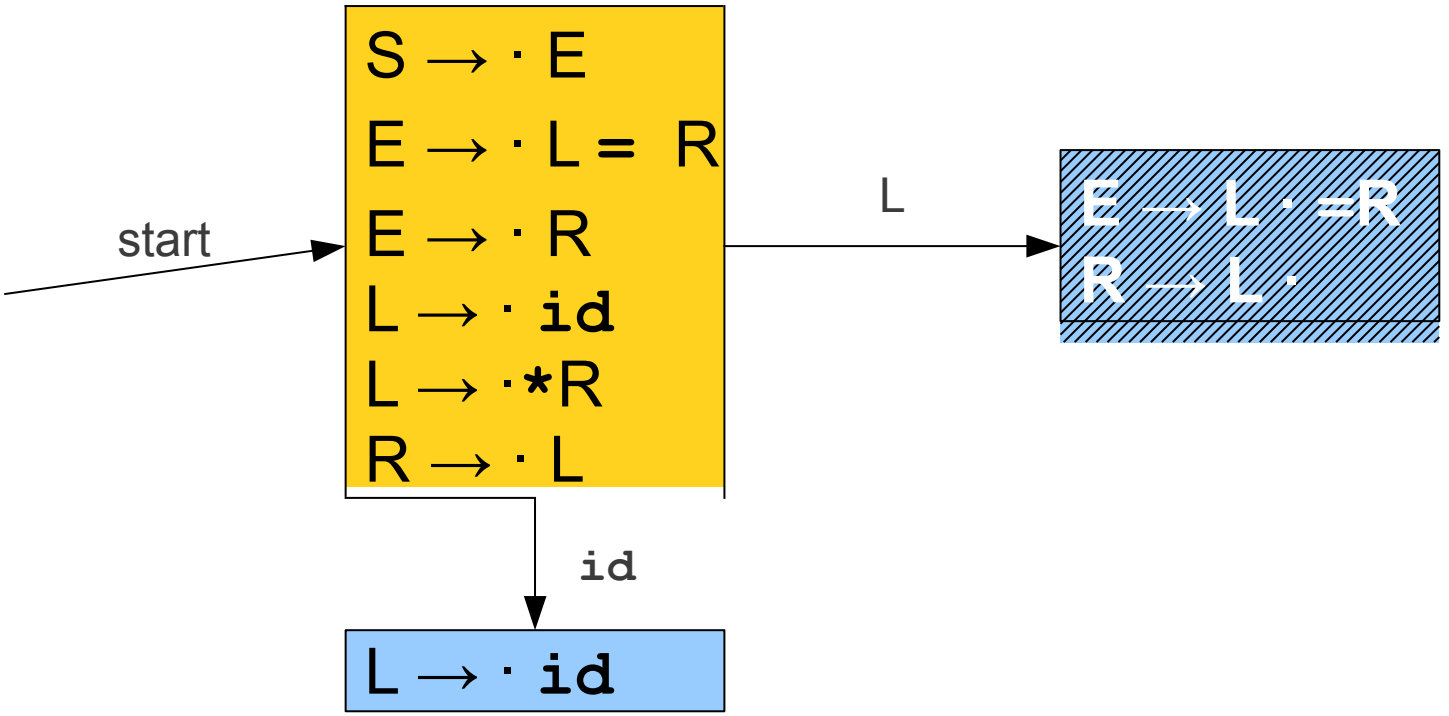
S → E
E → L = R
E → R
L → id
L → *R
R → L



id	=	*	id

A Lack of Context

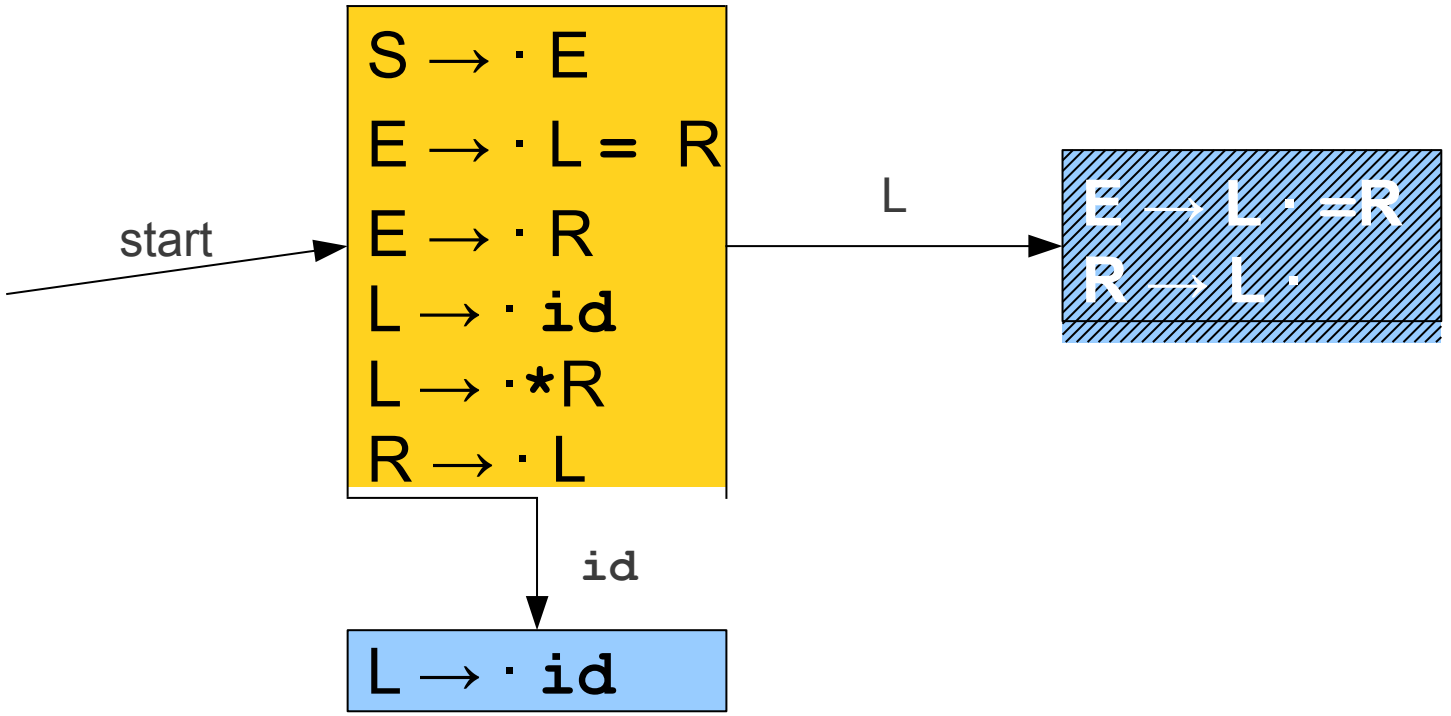
S → E
E → L = R
E → R
L → id
L → *R
R → L



id	=	*	id

A Lack of Context

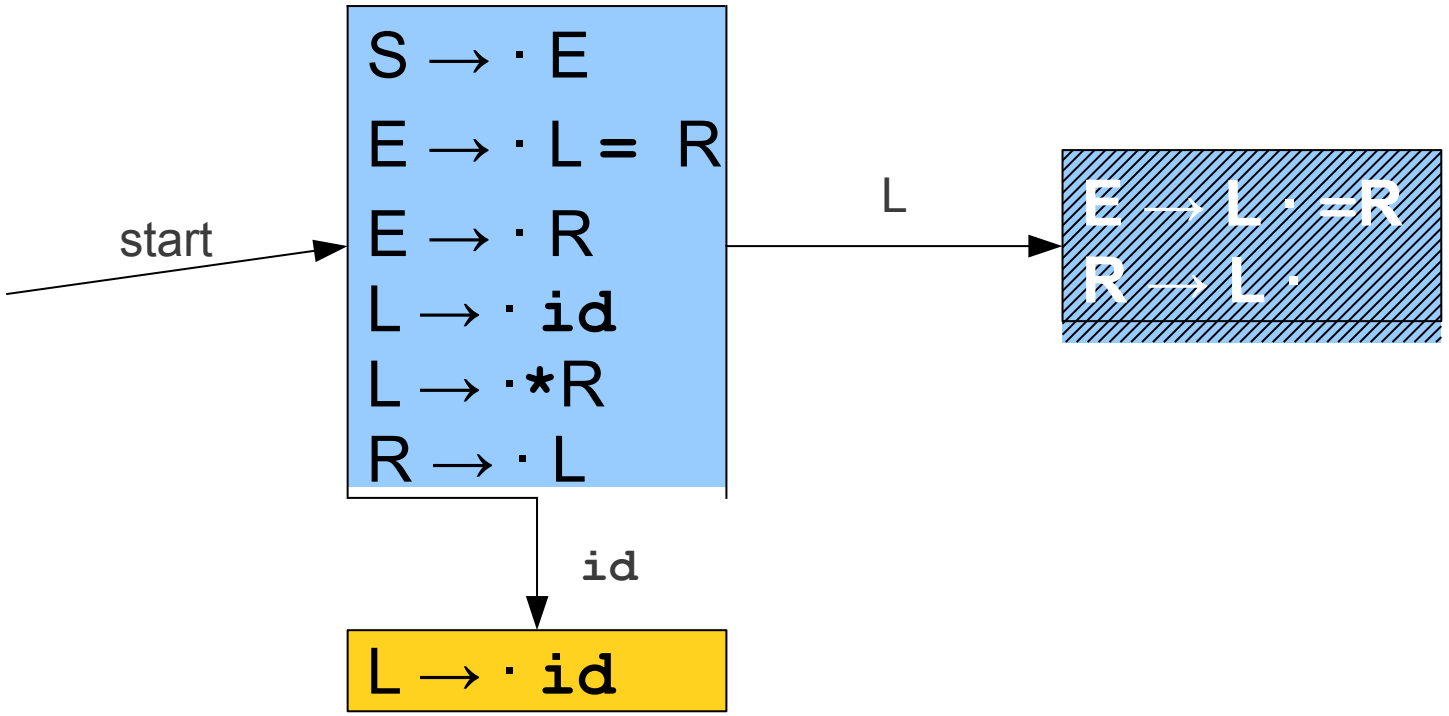
S → E
E → L = R
E → R
L → id
L → *R
R → L

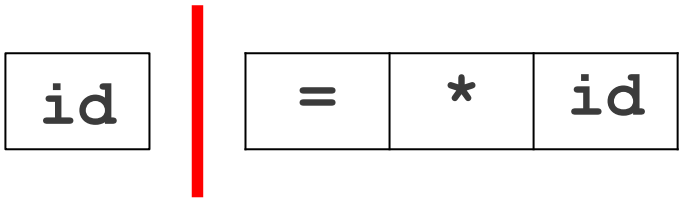


id	=	*	id

A Lack of Context

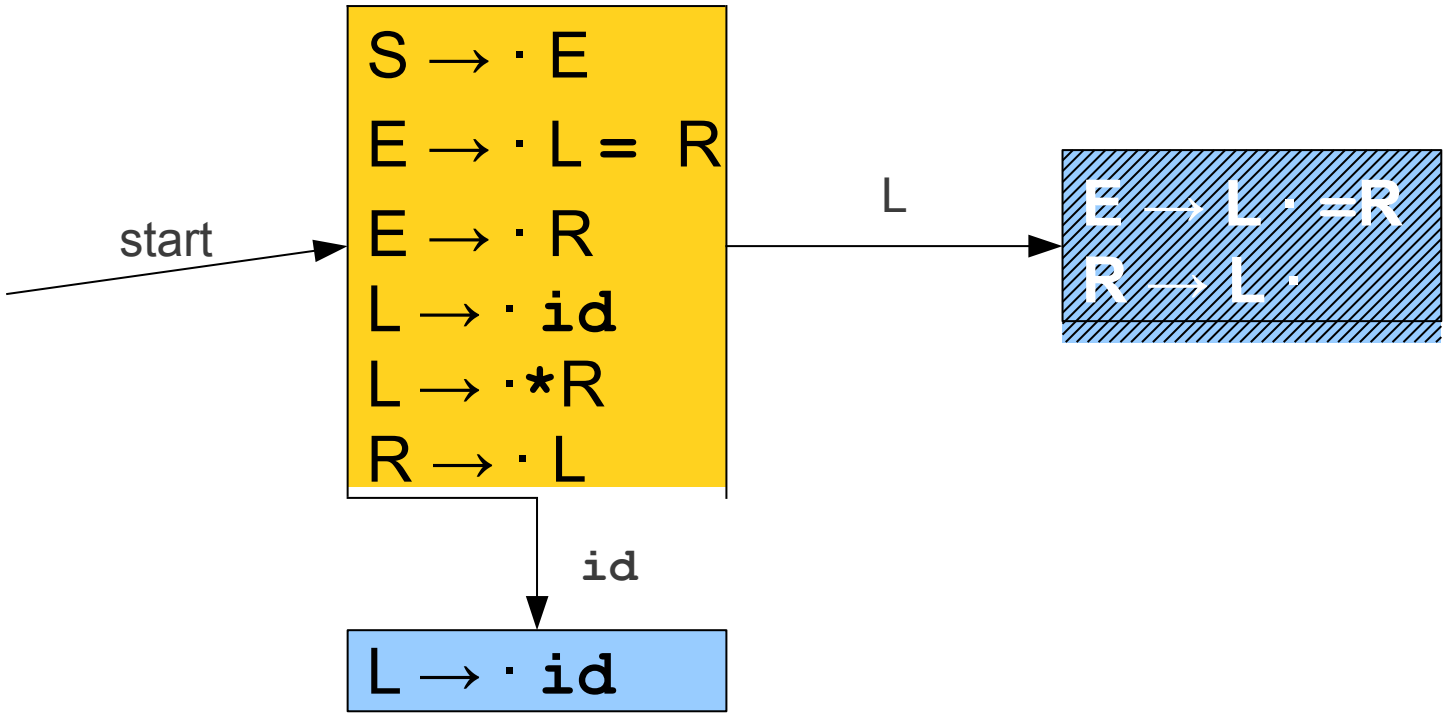
S → E
E → L = R
E → R
L → id
L → *R
R → L





A Lack of Context

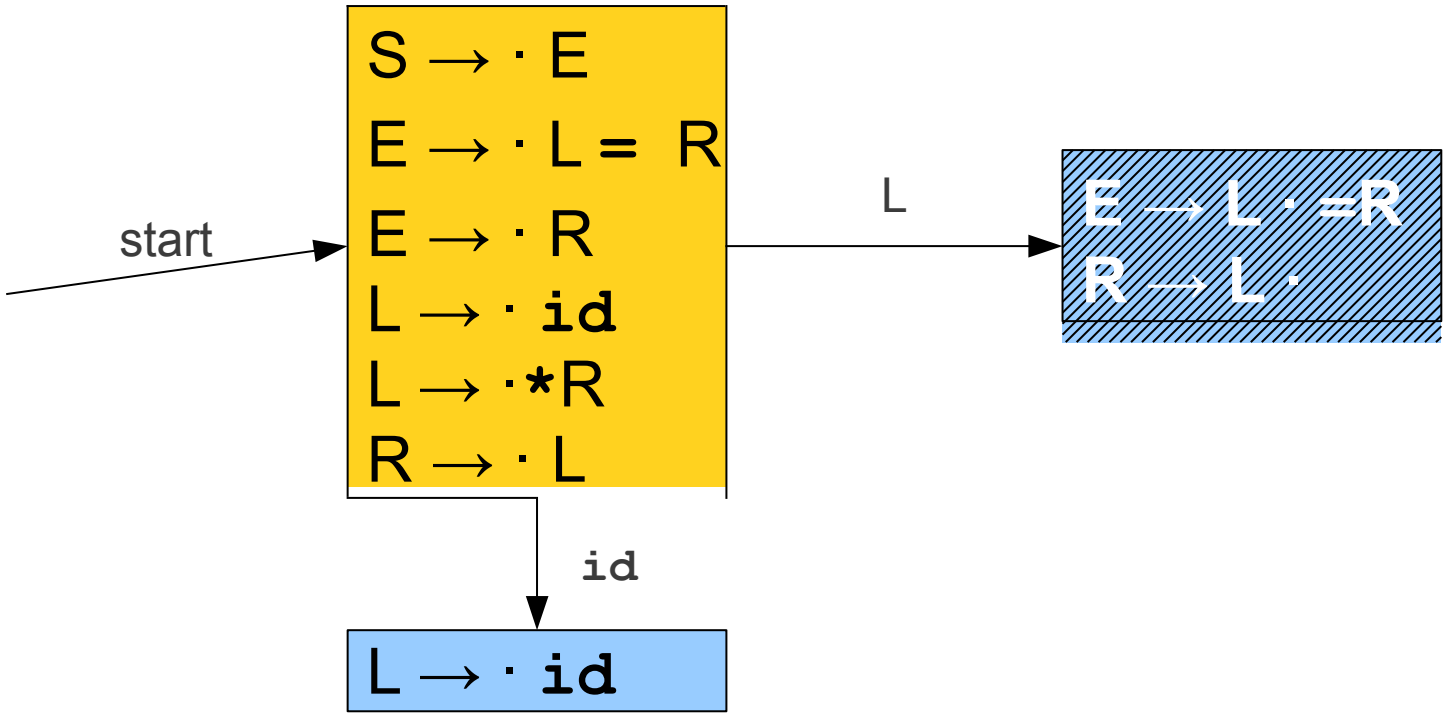
S → E
E → L = R
E → R
L → id
L → *R
R → L



=	*	id

A Lack of Context

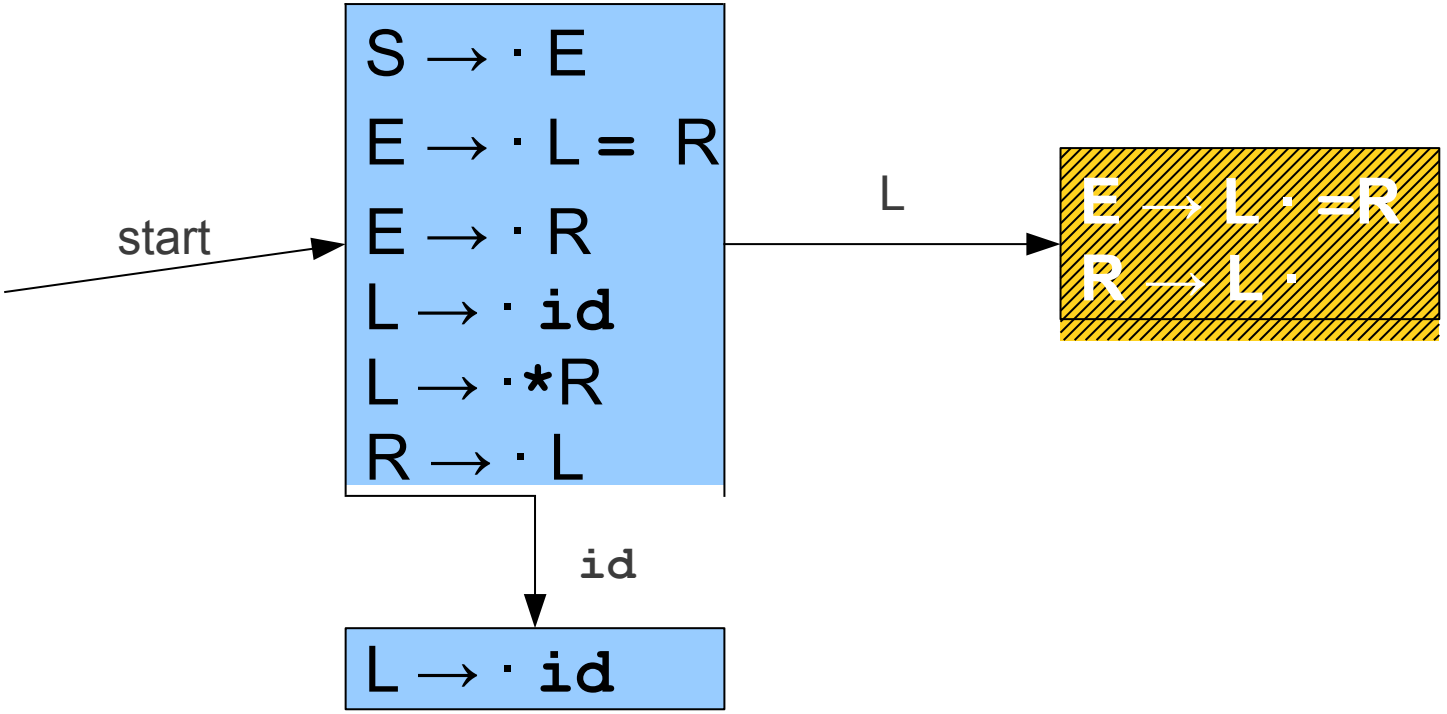
S → E
E → L = R
E → R
L → id
L → *R
R → L



L	=	*	id

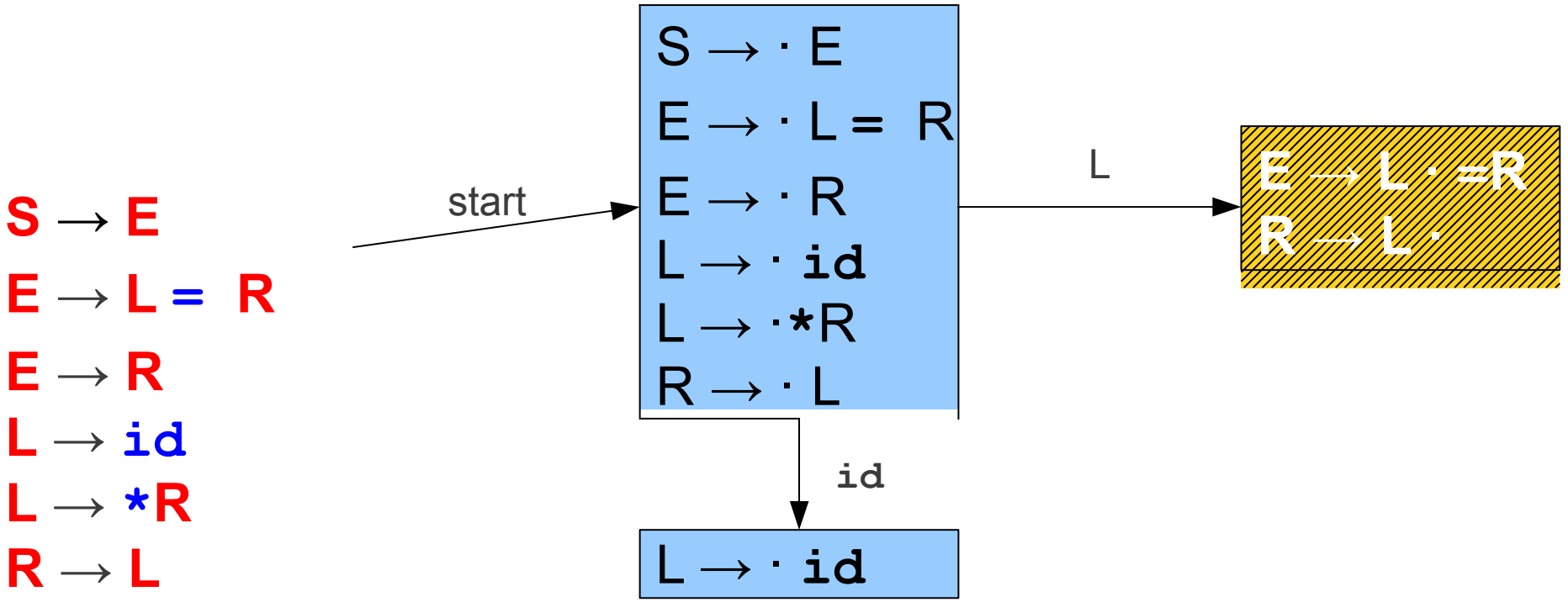
A Lack of Context

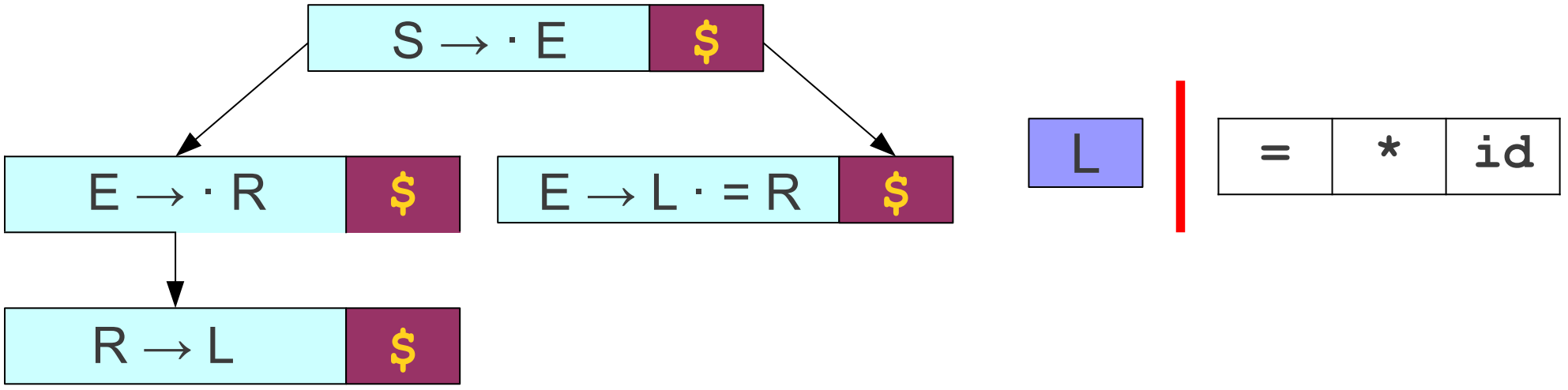
S → E
E → L = R
E → R
L → id
L → *R
R → L



L	=	*	id

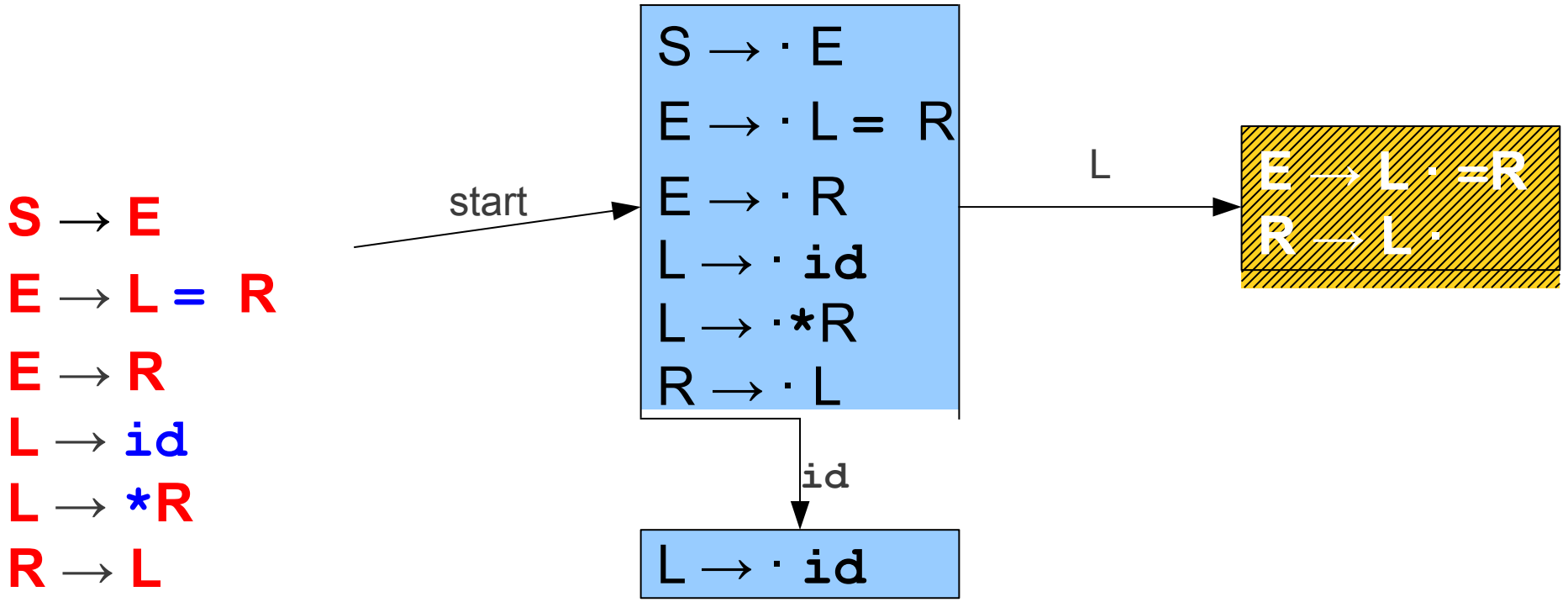
A Lack of Context

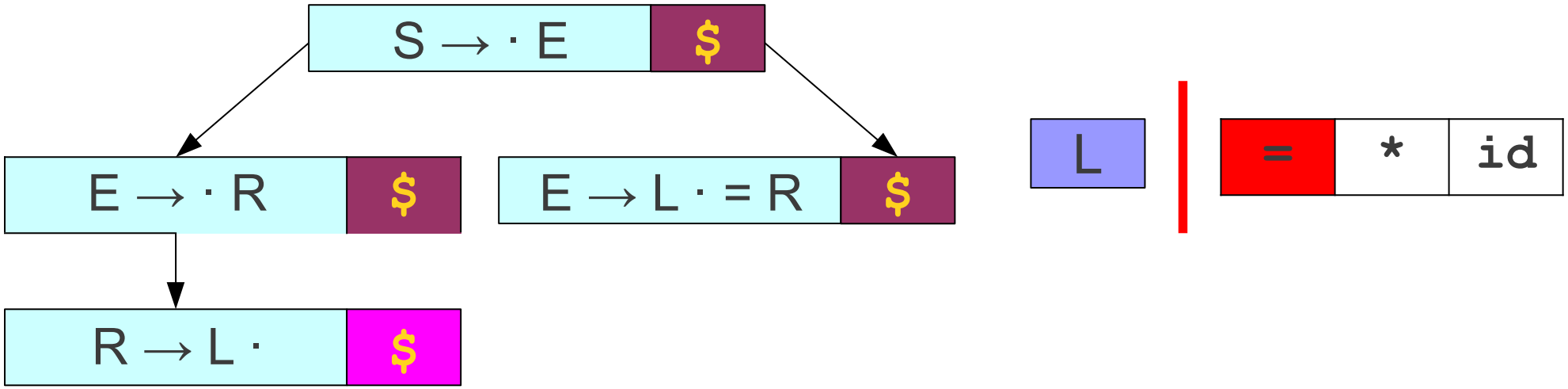




L	=	*	id

A Lack of Context





Why is SLR(1)Weak?

- With SLR(1), *minimal* context.
  - FOLLOW(**A**) means “what could follow **A** *somewhere* in the grammar?,” even if in a particular state **A** couldn't possibly have that symbol after it.
- With LR(1), we have contextual information.

DFA for LR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L

start

S_0

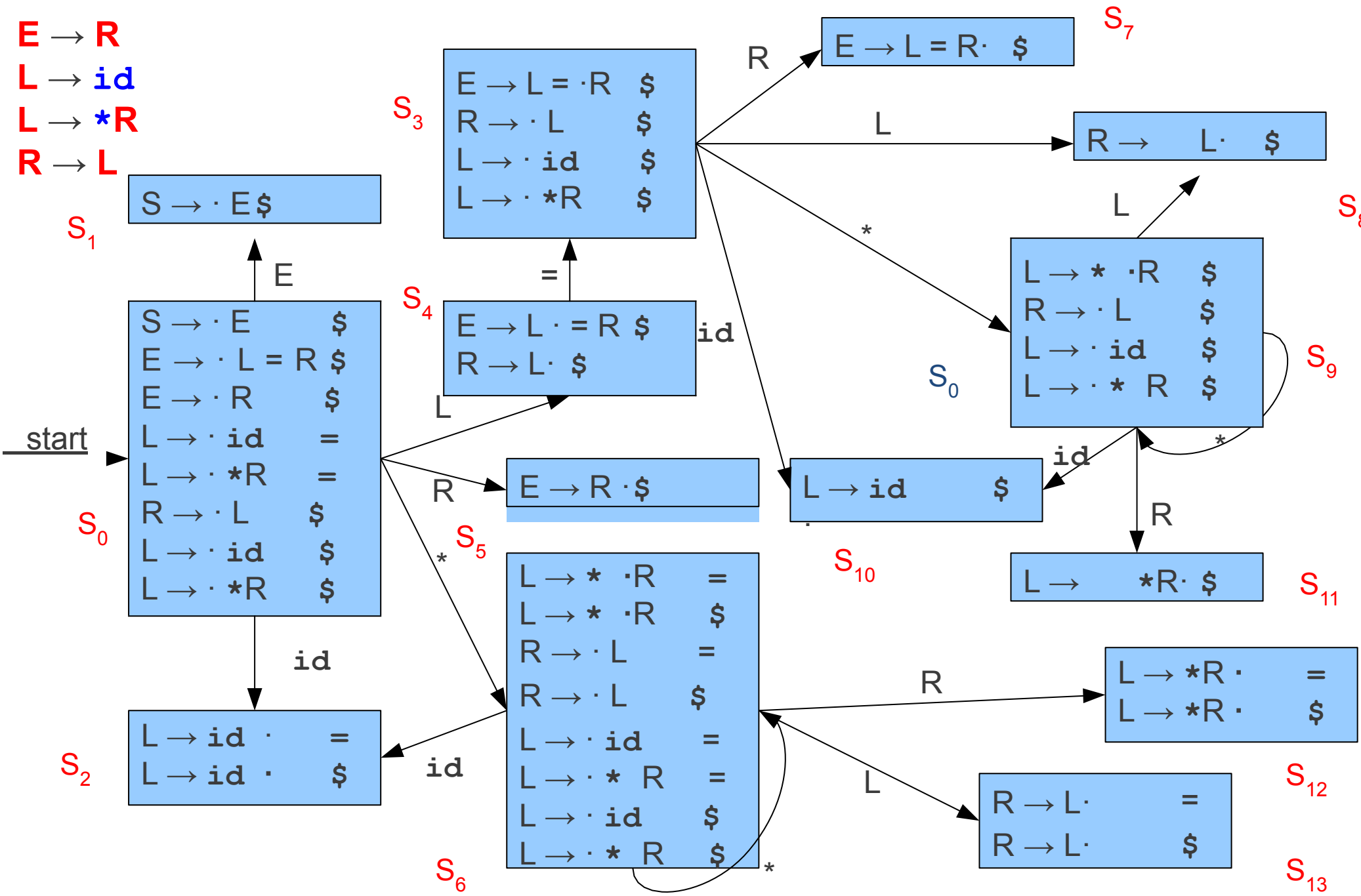
S → · E	$
E → · L = R	$
E → · R	$
L → · id	=
L → · *R	=
R → · L	$
L → · id	$
L → · *R	$

Constructing LR(1) Items

- Begin in a state containing **S** → · **E** , **$** where **S** is the start symbol and **$ is lookahead**
- Compute the **closure** of the state:
  - If **A** → *a* · **B***ω*, ***l*** is in the state, add **B** → · *γ*, ***t*** to the state for each production **B** → *γ* and for each terminal **t** ∈ FIRST*(*ω***l**)

DFA for LR(1)

S → E
E → L = R
E → R
L → id
L → *R
R → L



The LR(1) Parsing Algorithm

- Begin with an empty stack and the input set to ω**$**, where ω is the string to parse. Set **state** to the initial state.
- Repeat the following:
  - Let the next symbol of input be **t**.
  - If **action[state, t]** is **shift**, then shift the input and set **state** = **goto[state, t]**.
  - If **action[state, t]** is **reduce** **A** $\rightarrow$ ω:
    - Pop $|\omega|$ symbols off the stack; replace them with **A**.
    - Let the state atop the stack be **top-state**.
    - Set **state** = **goto[top-state, A]**
  - If **action[state, t]** is **accept**, then the parse is done. If
  - **action[state, t]** is **error**, report an error.

Constructing LR(1) Parse Tables

- For each state *X*:
  - If there is a production **A** → ω · [**t**], set **action**[*X*, **t**] = reduce **A** → ω.
  - If there is the special production **S** → **E** · [**$**], where **S** is the start symbol, set **action**[*X*, **t**] = **accept**.
  - If there is a transition out of s on symbol **t**, set **action**[*X*, **t**] = **shift**.
- Set all other actions to **error**.
- If any table entry contains two or more actions, the grammar is not LR(1).