NTNU – Trondheim
Norwegian University of
Science and Technology

SLR, LALR and LR(1) parsing tables

Limitations of LR(0)

- We have seen how LR parsing operates in terms of an automaton + a stack
  - States are created from closures of items
  - Transitions are actions based on the top of the stack, either before or after the next token is shifted
- The grammars that fit LR(0) are a bit more restrictive than they need to be
  - Specifically, they can stall on decisions which can easily be resolved by looking ahead in the token stream

To shift, or to reduce?

- Consider this grammar (which models arbitrarily long sums of terms)

S → E	(A statement is an expression)
E → T + E	(An expr. can be a sum of a term and an expr.)
E → T	(An expr. can be a term)
T → x	(A term can be a number, variable, whatever)

- The start symbol has just one production, we won't need to augment the grammar with any placeholder

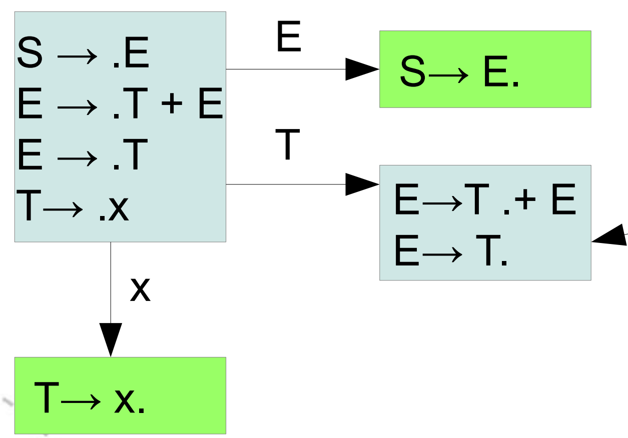
$S \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow x$

In short order

- Closure of $S \rightarrow .E$ is a state

$S \rightarrow .E$
$E \rightarrow .T + E$
$E \rightarrow .T$
$T \rightarrow .x$

- Transitions on E, T, x, find closures at destination:

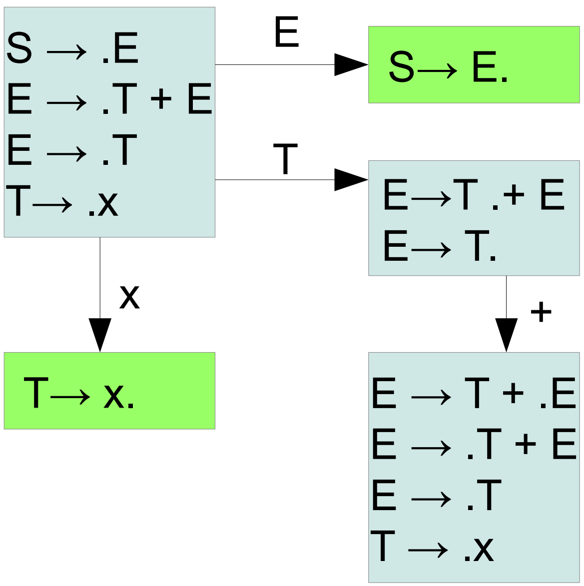


Whoops, there is a reduction here

S → E
E → T + E
E → T
T → x

In short order

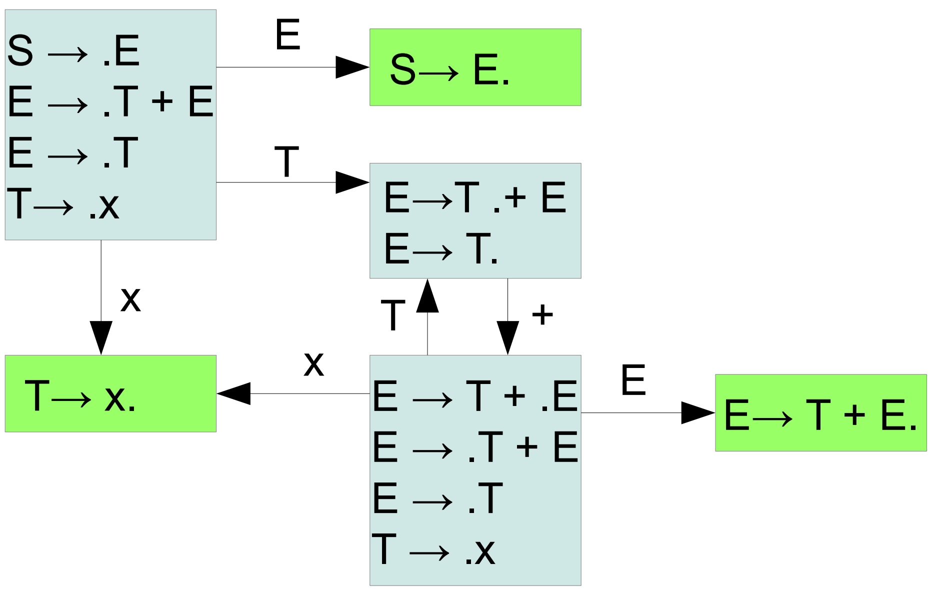
- Transition on +, find closure at destination



S → E
E → T + E
E → T
T → x

In short order

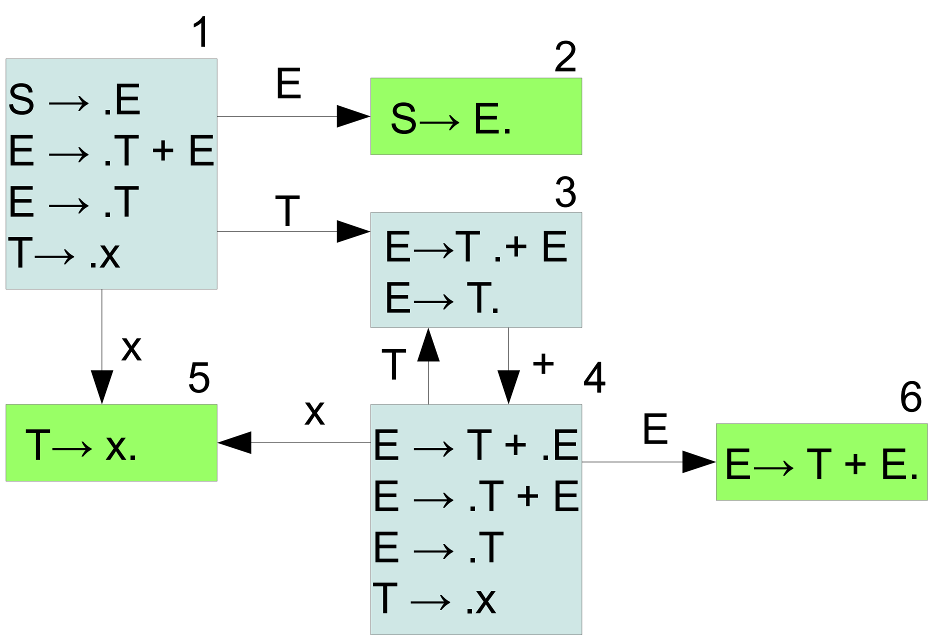
- Transitions on T, E, x, closures, and we're done



0) S → E
1) E → T + E
2) E → T
3) T → x

Numbers everywhere

- In the grammar, and on the states

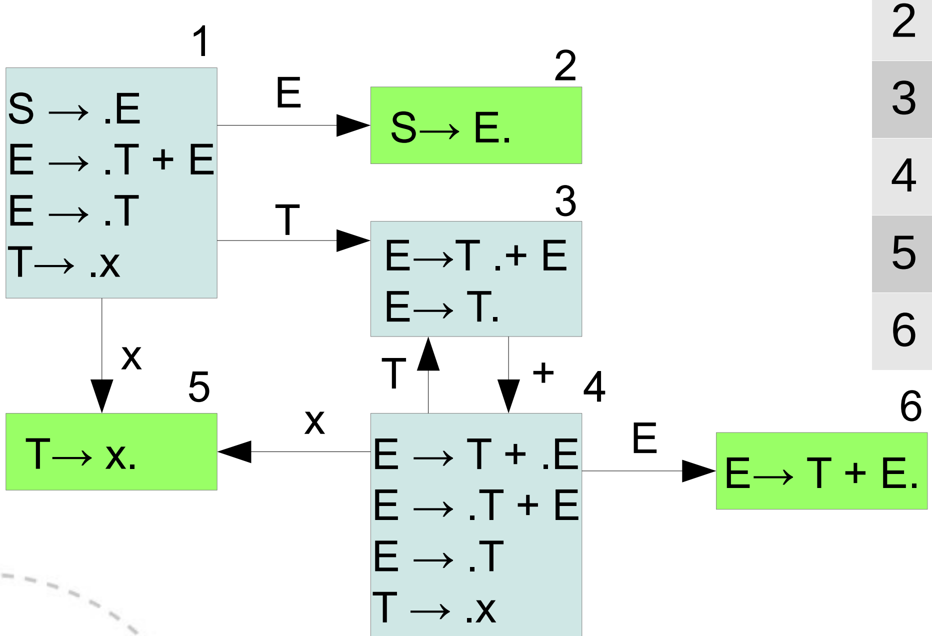


0) S → E
1) E → T + E
2) E → T
3) T → x

Most of the LR(0) table

- Here's what we get for the unproblematic states:

	x	+	$	E	T
1	s5			g2	g3
2			a		
3					
4	s5			g6	g3
5	r3	r3	r3		
6	r1	r1	r1		

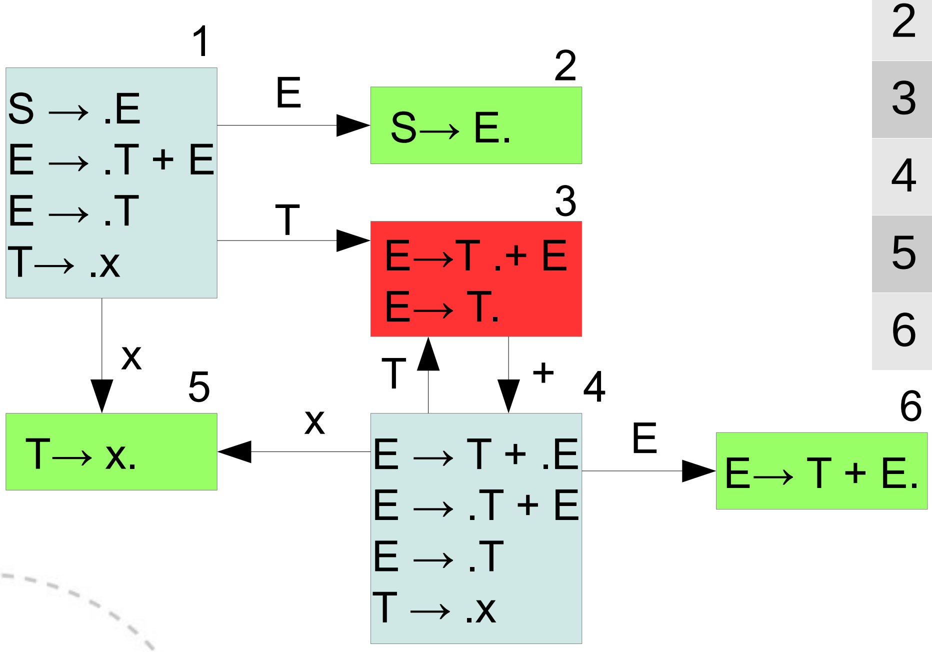


0) S → E
1) E → T + E
2) E → T
3) T → x

Shift/reduce conflict

- State 3 could shift and go to 4 on '+'

	x	+	$	E	T
1	s5			g2	g3
2			a		
3		s4			
4	s5			g6	g3
5	r3	r3	r3		
6	r1	r1	r1		

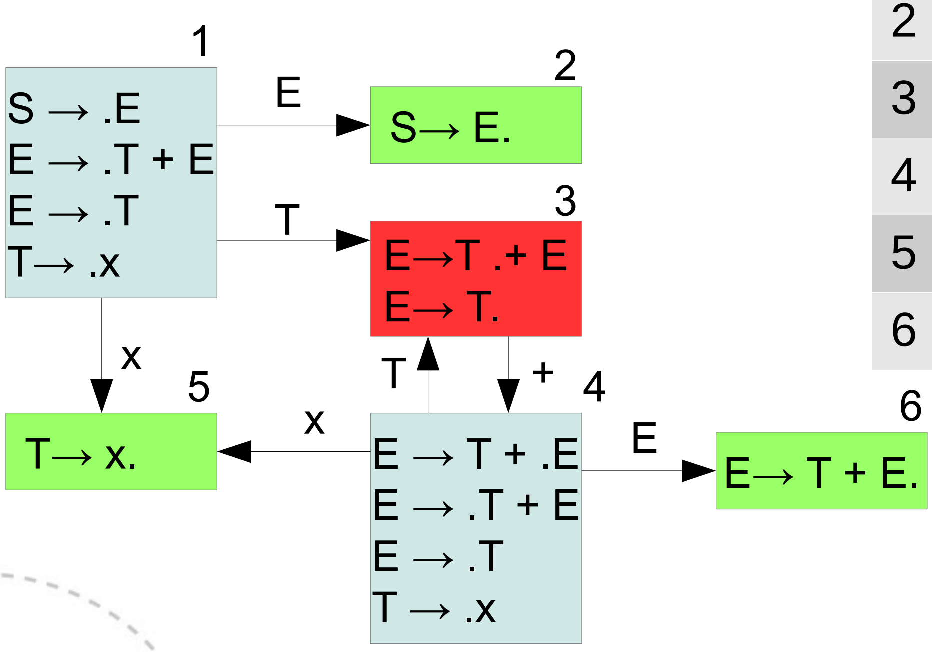


0) S → E
1) E → T + E
2) E → T
3) T → x

Shift/reduce conflict

- State 3 could also reduce production 2
- Parser can't decide here

	x	+	$	E	T
1	s5			g2	g3
2			a		
3	r2	r2,s4	r2		
4	s5			g6	g3
5	r3	r3	r3		
6	r1	r1	r1		



The immediate solution

- Wait with reductions until there are no more + tokens to shift
  - Like the longest match rule for regex
- All we need to know is what the next token will be
  - Buffer it, to look at what's coming
- When are we interested?
  - When the next token belongs to a construct that only comes after the nonterminal we are working through a production for
- We did that already
  - For a production $A \rightarrow \alpha$, any expected token which isn't in α goes into the set of tokens FOLLOW(A)
  - That is its definition

Reworking the reductions

- With 1 token lookahead, reducing states no longer need to reduce regardless of what comes next
- We can insert reduce actions a little more selectively, that is

  When an item A→α. suggests that a state is reducing,

  put the reducing action in the table only at tokens in FOLLOW(A)

NTNU – Trondheim
Norwegian University of
Science and Technology

0) $S \rightarrow E$
1) $E \rightarrow T + E$
2) $E \rightarrow T$
3) $T \rightarrow x$

Reworking the reductions

- $E \rightarrow T.$ is our problem item here
  - FOLLOW(E) = {$} , by prod. 0; E always remains on the far right in derivations
- $E \rightarrow T + E.$ is a reduction, too
  - We already found FOLLOW(E)
- $T \rightarrow x.$

  FOLLOW(T) = {+,$} (+ because of prd. 1, $ because of prd. 2)
- $S \rightarrow E.$

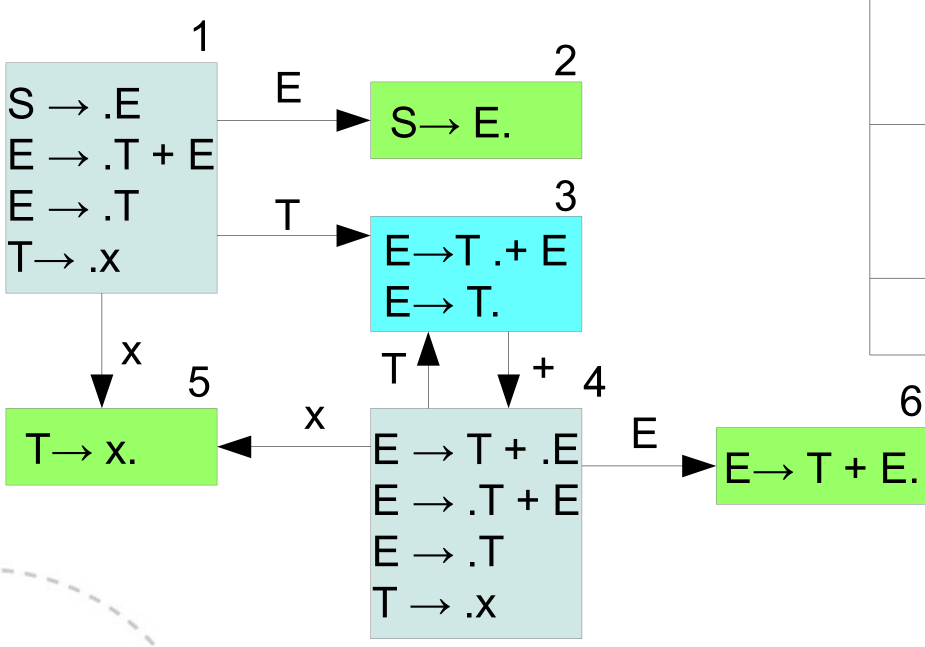
  FOLLOW(S) = {$} (S is never on a r.h.s of anything)

0) S → E
1) E → T + E
2) E → T
3) T → x

An updated table

- Taking this into account, state 3 is no longer difficult
- Changes affect these rows

	x	+	$	E	T
1	s5			g2	g3
2			a		
3		s4	r2		
4	s5			g6	g3
5		r3	r3		
6			r1		



NTNU – Trondheim
Norwegian University of
Science and Technology

That was the SLR table

aka. "Simple LR"

- So named because it is just a tiny adjustment of the LR(0) scheme
- It does not, however, take all the information that it can out of having a lookahead symbol
- That's what the full-blown LR(1) scheme does

NTNU – Trondheim
Norwegian University of Science and Technology

S' → S
S → V = E
S → E
E → V
V → x
V → *E

A grammar that needs more

- To revamp the whole scheme with lookahead symbols, the idea of an *item* can be extended
- Take this (sub-)grammar of expressions, variables, and pointer dereference a la C:

S' → S	(Unique production to start with)
S → V = E	(Expr. can be assigned to variables)
S → E	(Expressions are statements)
E → V	(Variables are expressions)
V → x	(Variables can be identifiers)
V → * E	(Variables can be dereferenced pointer expressions)

(...and pointer expressions can have variables in them…)

- This is not SLR *(Can you figure out why not?)*

NTNU – Trondheim
Norwegian University of Science and Technology

Revisit the items

- LR(1) items include a lookahead symbol

  A → α . X β   says we're ahead of X between α and β

  A → α. X β   *t*   says the same, but t is the next token

- Take an item like [A → . X &   %]

  '%' might be found in some expansion of X, so we need

  X → . <*something*>   %

  X → . <*somethingelse*>   %

  and all variants of X while foreseeing '%'.

- It can also be that X will reduce without shifting more stuff

  The production says that we might see '&' as lookahead at this point, so

  X → . <*something*>   &

  X → . <*somethingelse*>   &

  are also possibilities we must include in the closure.

S' → S
S → V = E
S → E
E → V
V → x
V → *E

For our grammar

- Starting out as before, we get that

  S' → .S *?*

  has no sensible lookahead, because you can't look beyond the end
- After S comes $, carry that through all nonterminal expansions

S → .V = E	$
S → .E	$
E → .V	$
V → .x	$
V → .*E	$

S’ → S
S → V = E
S → E
E → V
V → x
V → *E

Are there other relevant lookaheads?

- Looking at

  S → .V = E

  it is possible that we’re about to go to work on a V, and there is an ‘=’ token coming up after it

- Taking it into account

  S → .V = E

  gives that

  V → .x =

  V → .*E =

  also belong in the closure of LR(1) items

(In excessive notation, include the item [X → α, ω] for ω in FIRST(βz) where the item you’re working out the closure for can be written [A → α.Xβ, z]…)

S' → S	
S → V = E	
S → E	
E → V	
V → x	
V → *E	

For short

- The first state of our LR(1) automaton thus becomes

S' → .S	?
S → .V = E	$
S → .E	$
E → .V	$
V → .x	$
V → .*E	$
V → .x	=
V → .*E	=

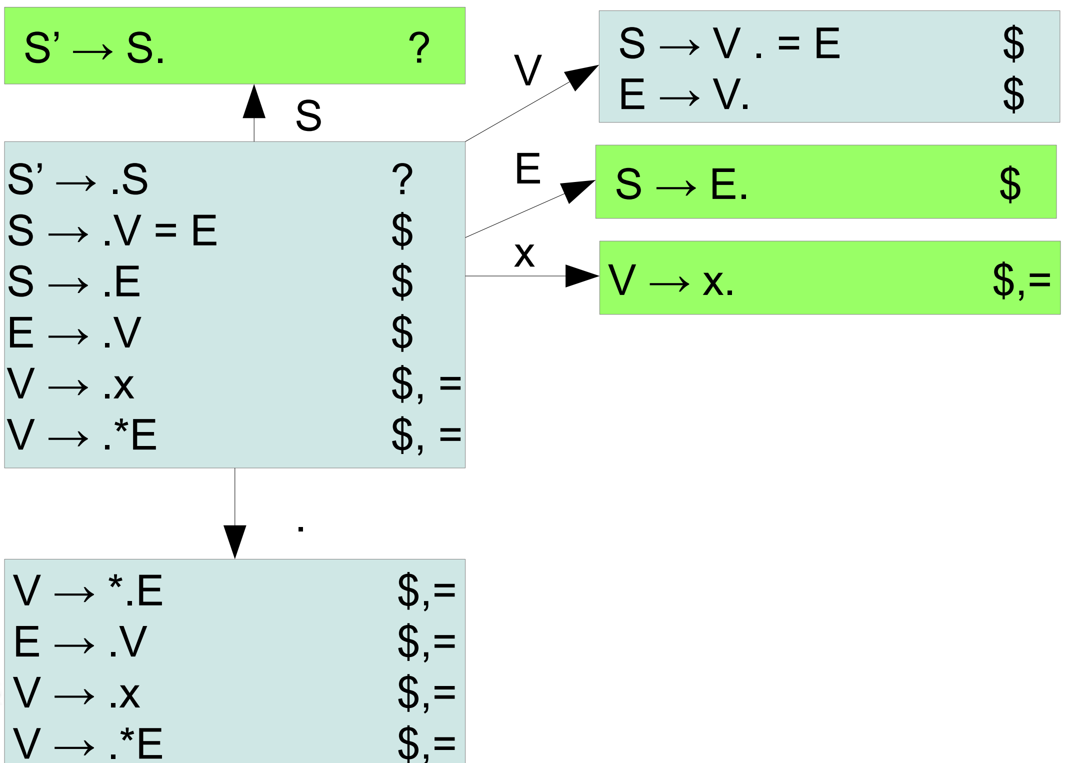
which we might as well write

S' → .S	?
S → .V = E	$
S → .E	$
E → .V	$
V → .x	$, =
V → .*E	$, =

S’ → S
S → V = E
S → E
E → V
V → x
V → *E

Building the automaton

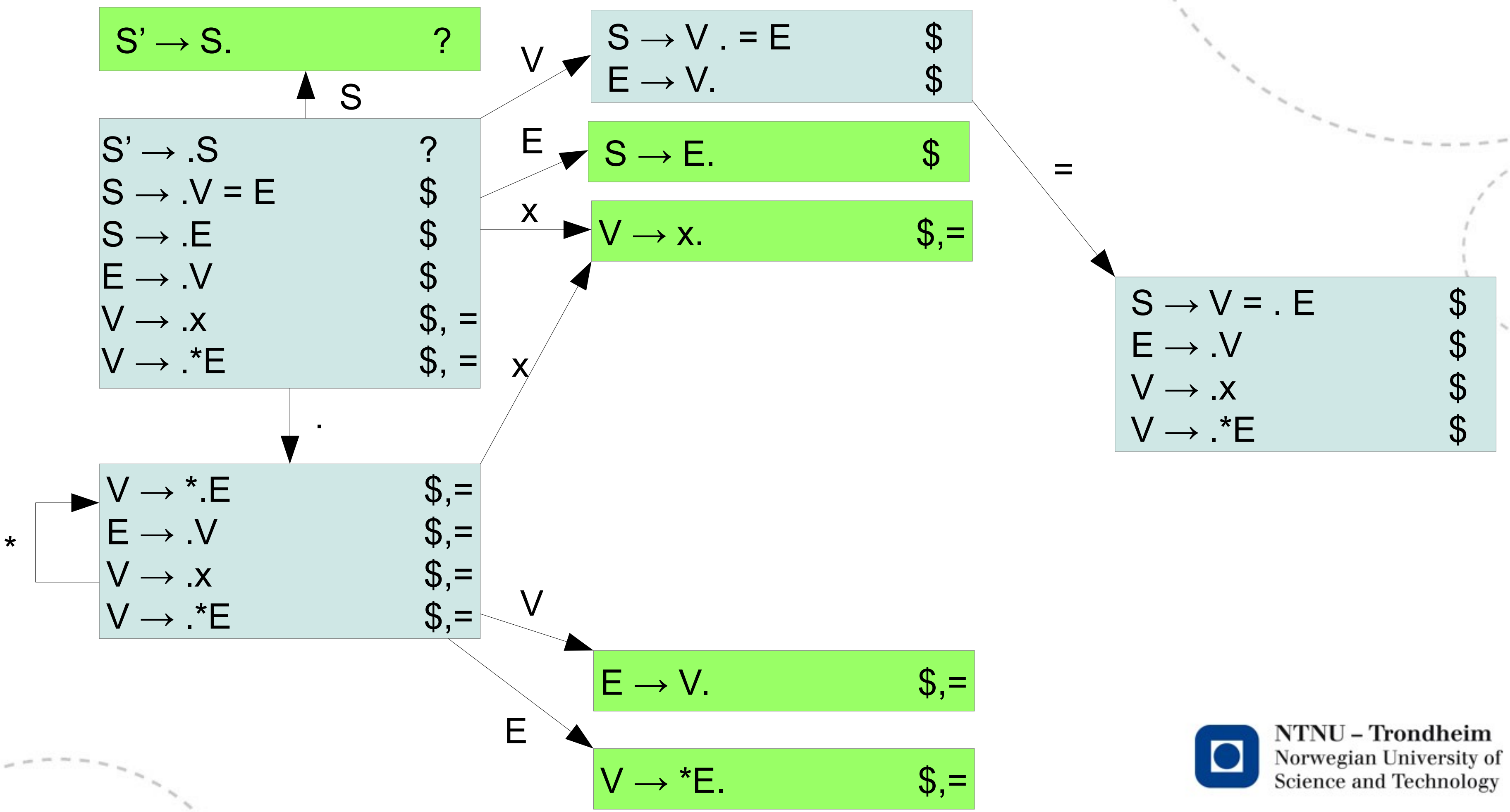
- The procedure remains the same, just with more elaborate closures



NTNU – Trondheim
Norwegian University of
Science and Technology

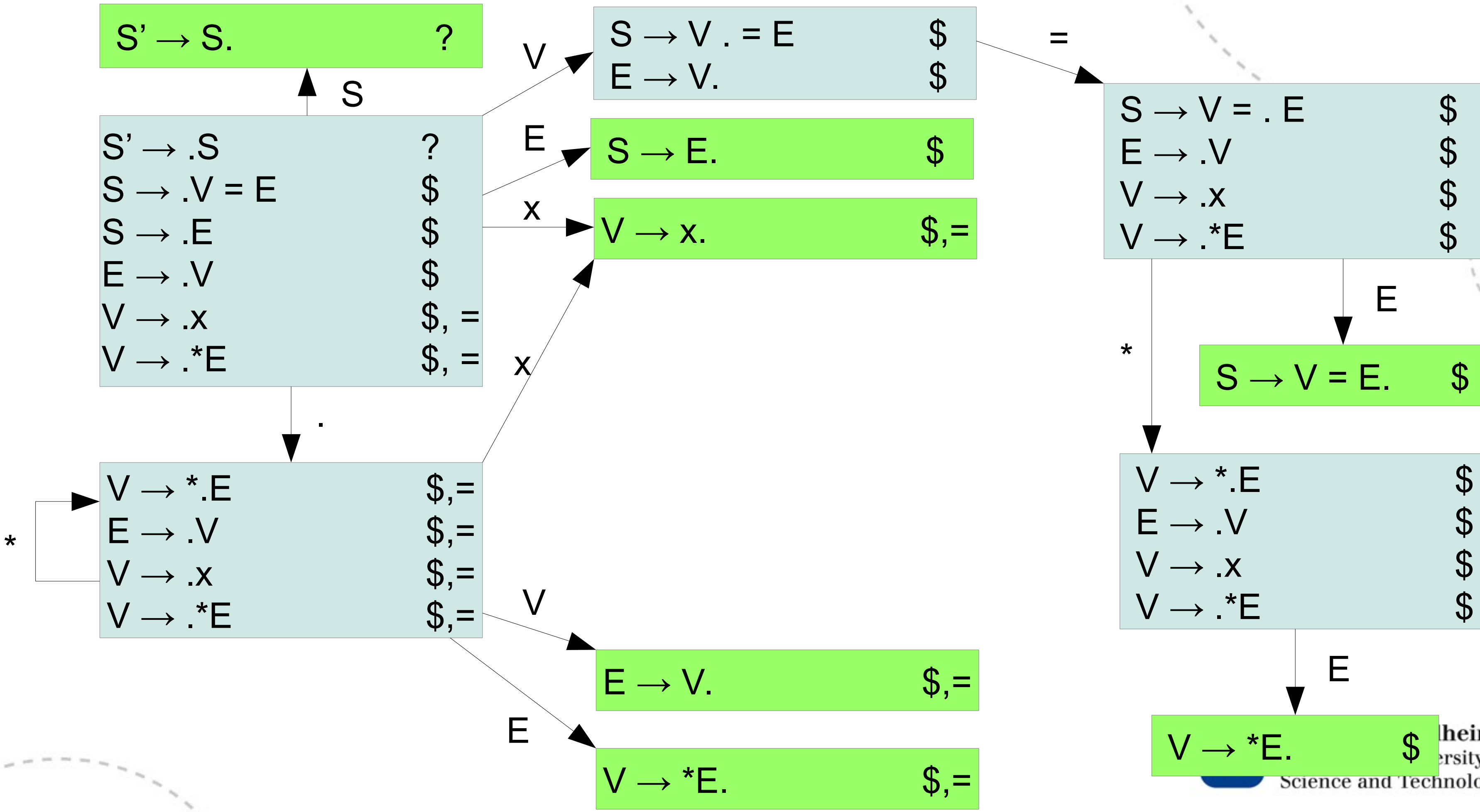
Building the automaton

S’ → S
S → V = E
S → E
E → V
V → x
V → *E



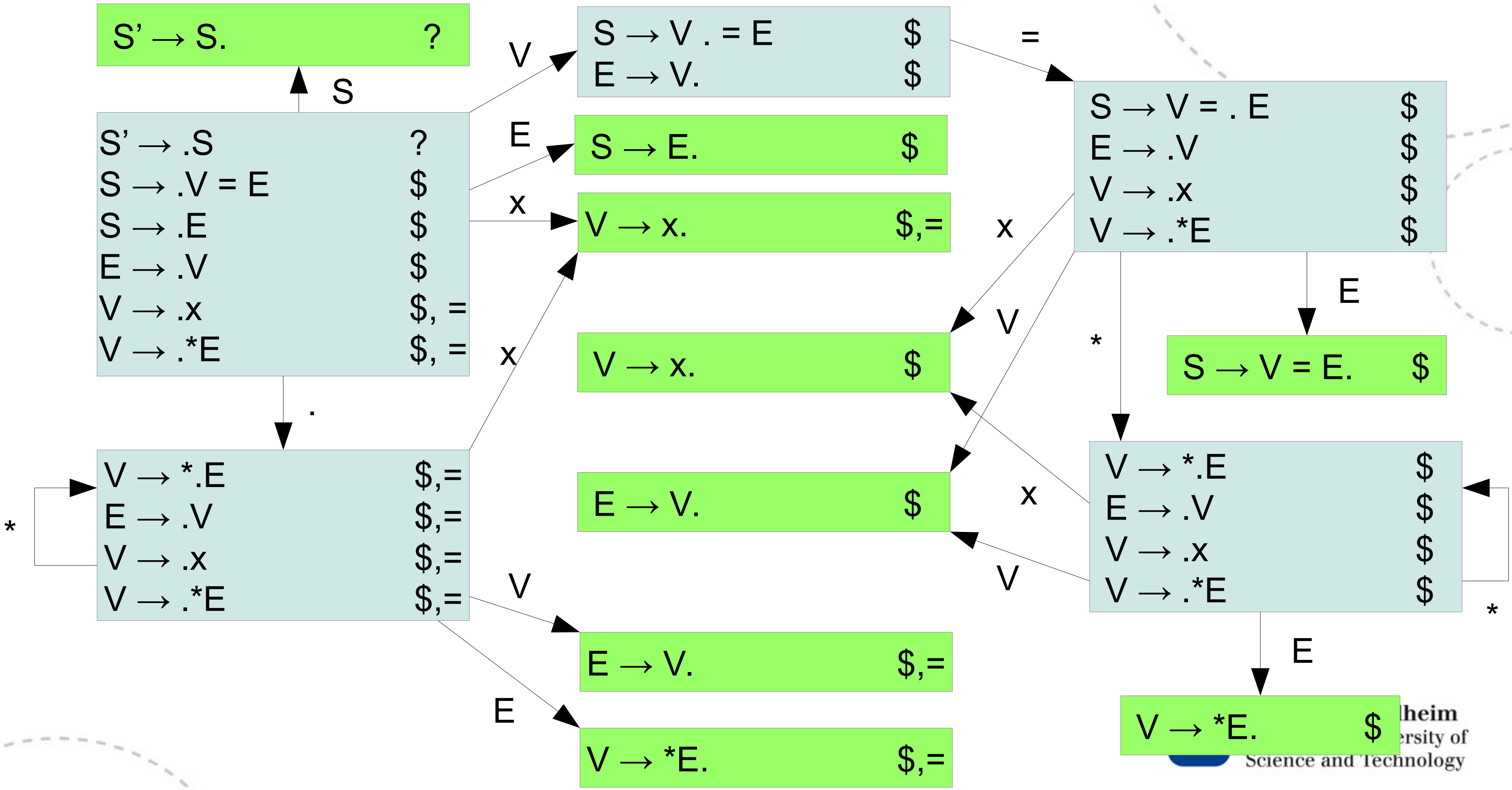
S' → S
S → V = E
S → E
E → V
V → x
V → *E

Building the automaton



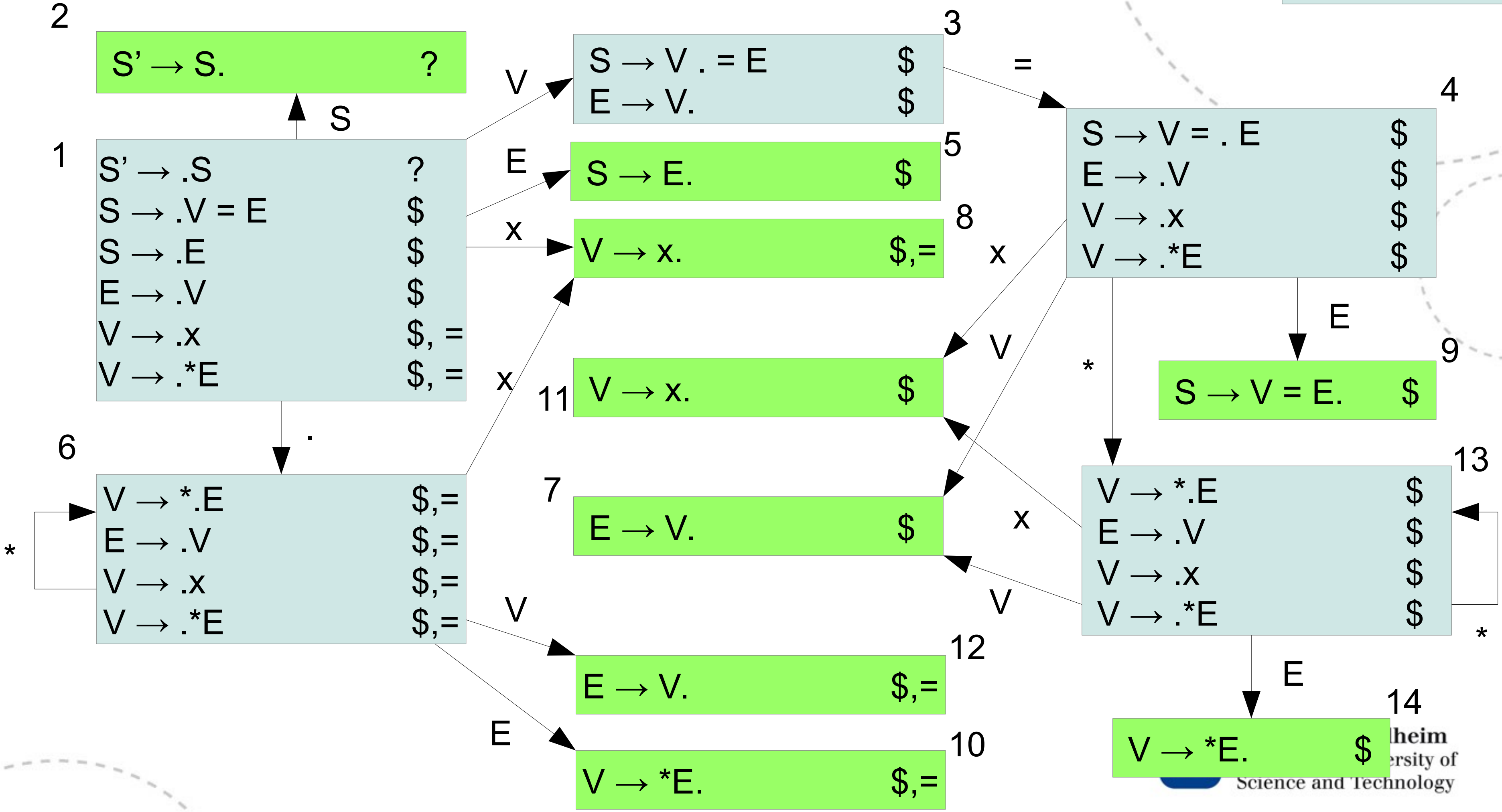
This is it

S' → S
S → V = E
S → E
E → V
V → x
V → *E



Number states & productions

0) S' → S
1) S → V = E
2) S → E
3) E → V
4) V → x
5) V → *E



Where to put reduce actions

- When an item reduces, its lookahead symbol decides where to tabulate the reduction
- That's the reason why we wanted to track lookahead symbols in the first place

0) S' → S
1) S → V = E
2) S → E
3) E → V
4) V → x
5) V → *E

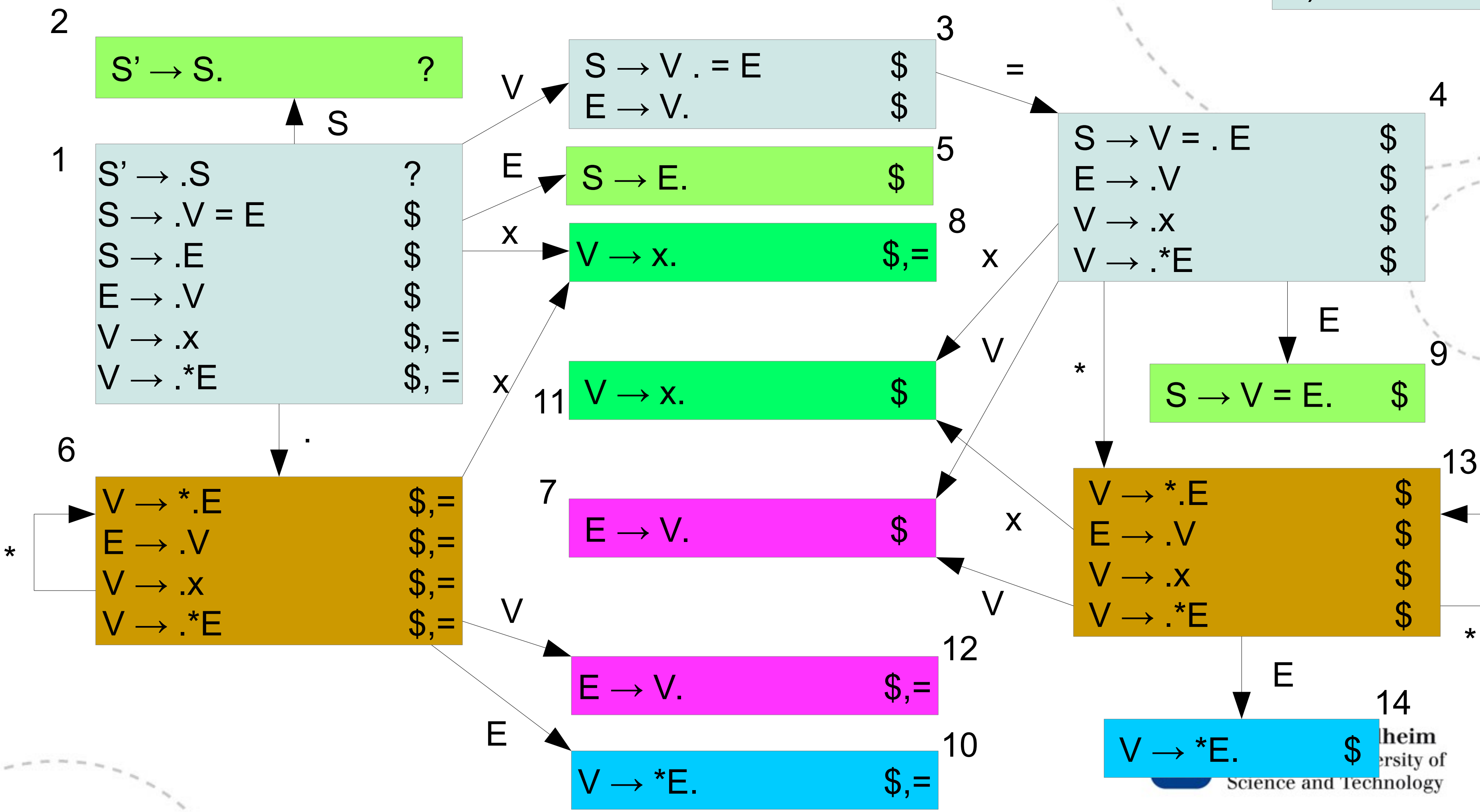
LR(1) parsing table

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

As you may notice

Some of these states are pretty similar...

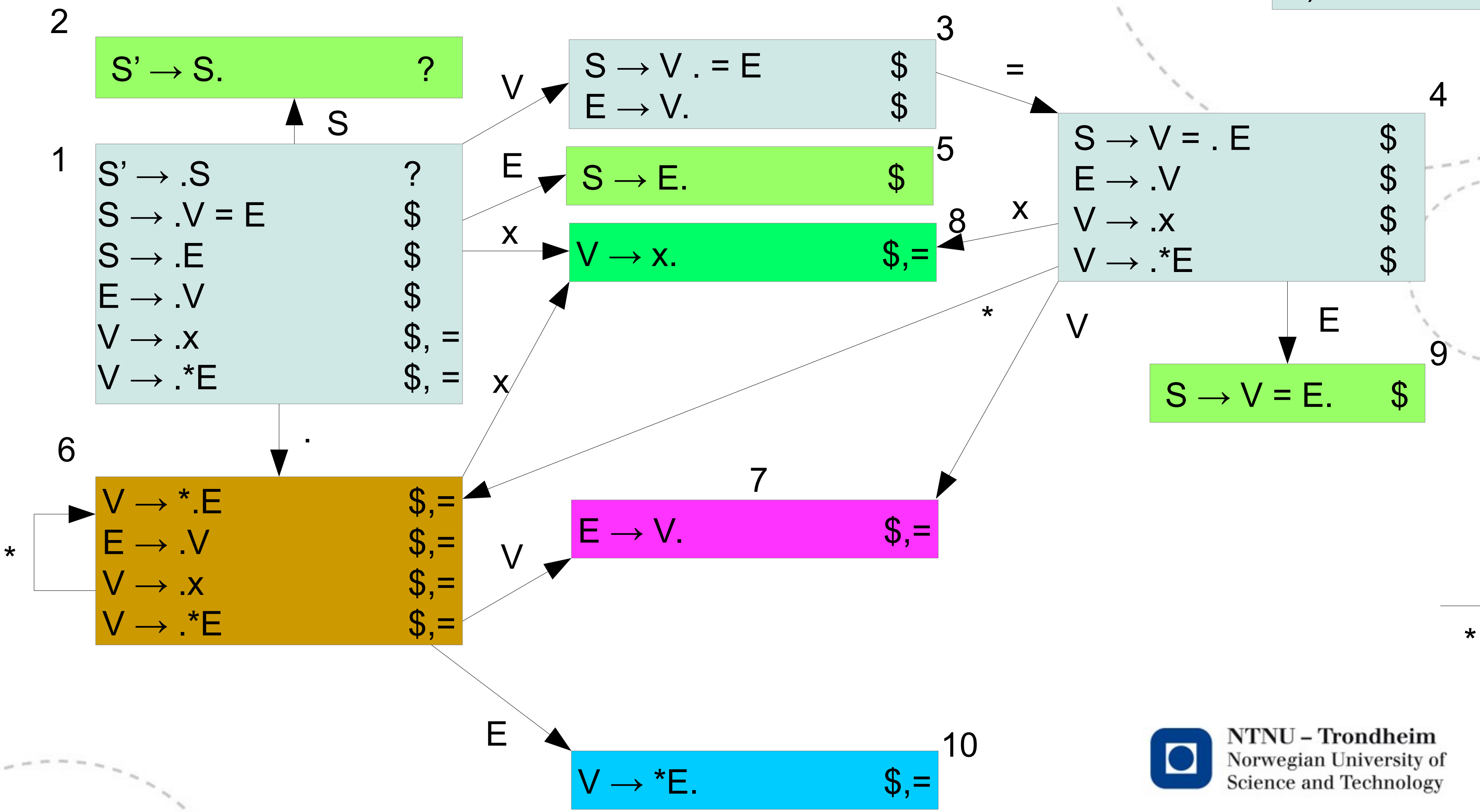
0) S’ → S
1) S → V = E
2) S → E
3) E → V
4) V → x
5) V → *E



What if we merge them?

i.e. those which are similar except for the lookahead

0) S' → S
1) S → V = E
2) S → E
3) E → V
4) V → x
5) V → *E



0) S' → S
1) S → V = E
2) S → E
3) E → V
4) V → x
5) V → *E

LALR parsing table

LR parsing + this state reduction is Look-Ahead LR (LALR)

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s8	s6				g9	g7
5				r2			
6	s8	s6				g10	g7
7			r3	r3			
8			r4	r4			
9				r1			
10			r5	r5			