Canonical LR(1)
&
LALR(1) Parser

Definition of LR(1)

- Two-component element of the form

  $[A \to \alpha.\beta, u]$

  where 1st component is marked production

  $A \to \alpha.\beta$,called the core of the item

  u is a lookahead character belongs to the set $V_T \cup \{\varepsilon\}$.

Validity

- An LR(1) item $[A \rightarrow \alpha.\beta, u]$ is *valid* for *viable prefix* λ, *if there exists a rightmost derivation*

$$S \xrightarrow[R]{*} \Phi A t \xrightarrow[R]{} \Phi\alpha\beta t$$

Where $\lambda = \Phi\alpha$ is the viable prefix and u is the 1st symbol of t,or ε if t=ε.

Example

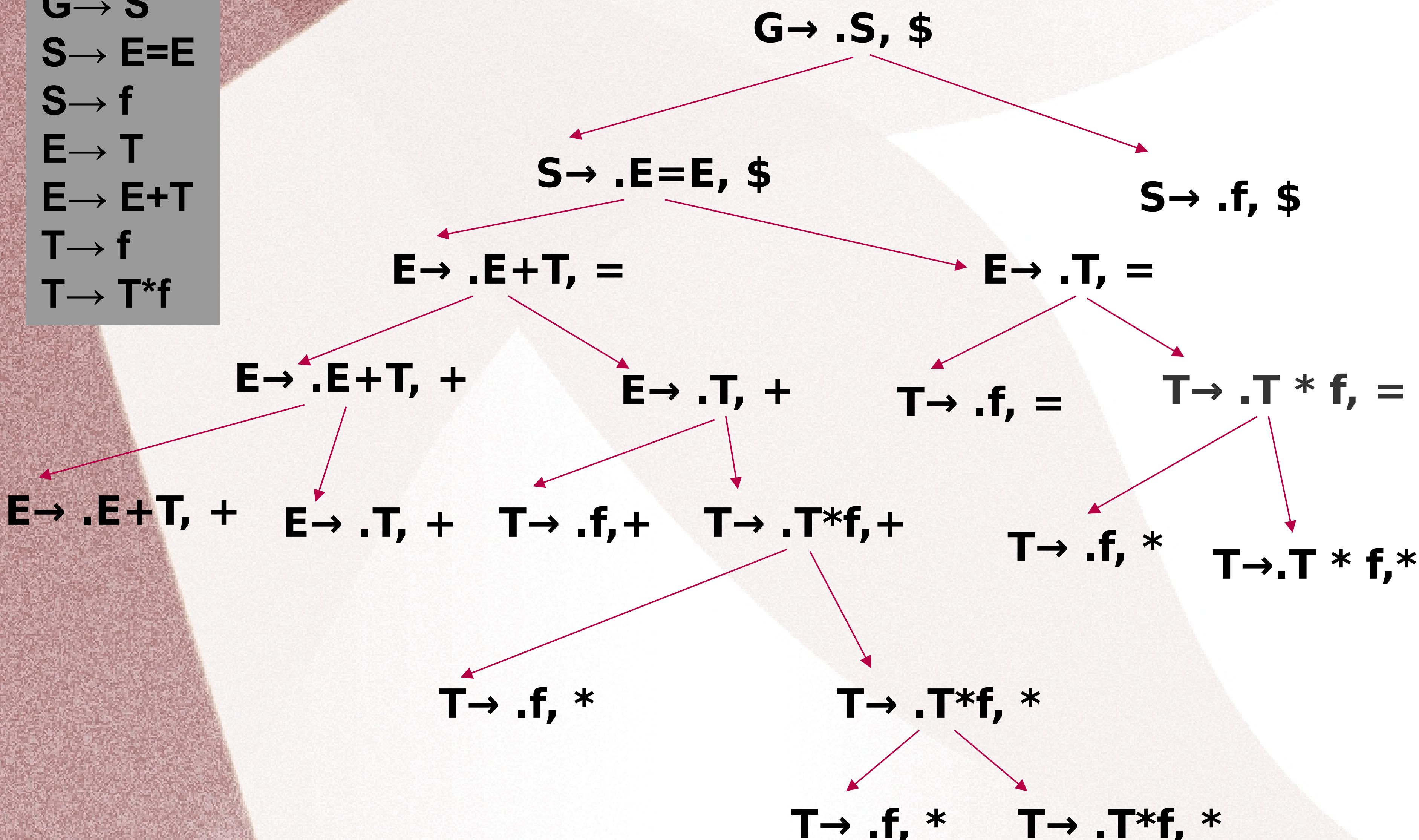
G→ S
S→ E=E
S→ f
E→ T
E→ E+T
T→ f
T→ T*f

State-0 (I0) :

G→ .S , $
S→ .E=E , $
S→ .f , $
E→ .T , = +
E→ .E+T , = +
T→ .f , + * =
T→ .T*f , + * =

Closure of a LR(1) example

$G \rightarrow S$
$S \rightarrow E=E$
$S \rightarrow f$
$E \rightarrow T$
$E \rightarrow E+T$
$T \rightarrow f$
$T \rightarrow T*f$



Example

G→ S
S→ E=E
S→ f
E→ T
E→ E+T
T→ f
T→ T*f

State-0 (I0) :

G→ .S , $
S→ .E=E , $
S→ .f , $
E→ .T , = +
E→ .E+T , = +
T→ .f , + * =
T→ .T*f , + * =

Example

<u>State0 (I0) :</u>

G→ .S , $
S→ .E=E , $
S→ .f , $
E→ .T , = +
E→ .E+T , = +
T→ .f , + * =
T→ .T*f , + * =

<u>State1 (I1) : from state 0 on S</u>

G→S. , $

<u>State2 (I2) : from state 0 on E</u>

S→ E. = E , $

<u>State3 (I3) : from state 0 on f</u>

S→f. , $
T→f. , = + *

Example

State4 (I4) : from state 0 on T

E→ T. , = +

T→ T.*f , + * =

State5 (I5) : from state 2 on =

S→E=.E , $

E→ .T , $ +

T→.f , $ + *

T→.T*f , $ + *

E→ .E+T , $ +

State6 (I6) : from state 2 on +

E→E+.T , = +

T→.f , = + *

T→.T*f , = + *

State7 (I7) :
From state 4 on *

T→T*.f , = + *

Example

<u>State8 (I8) :</u>
<u>From 5 on E</u>

$S \rightarrow E=E. , \$$

$E \rightarrow E.+T , \$ +$

<u>State9 (I9) :</u>
<u>From state 5 on T</u>

$E \rightarrow T. , \$ +$

$T \rightarrow T.*f , \$ + *$

<u>State10 (I10) :</u>
<u>From state 5 on f</u>

$T \rightarrow f. , \$ + *$

<u>State11 (I11) :</u>
<u>From state 6 on T</u>

$T \rightarrow T.*f , = + *$

$E \rightarrow E+T. , = +$

<u>State12 (I12) :</u>
<u>From state 6 on f</u>

$T \rightarrow f. , = + *$

Example

<u>State13 (I13) :</u>
<u>From state 7 on f</u>

T→ T*f. , = + *

<u>State14 (I14) :</u>
<u>From state 8 on +</u>

E→E+.T, $ +
T→.T*f, $ + *
T→.f, $ + *

<u>State15 (I15) :</u>
<u>From state 9 on *</u>

T→ T*.f , $ + *

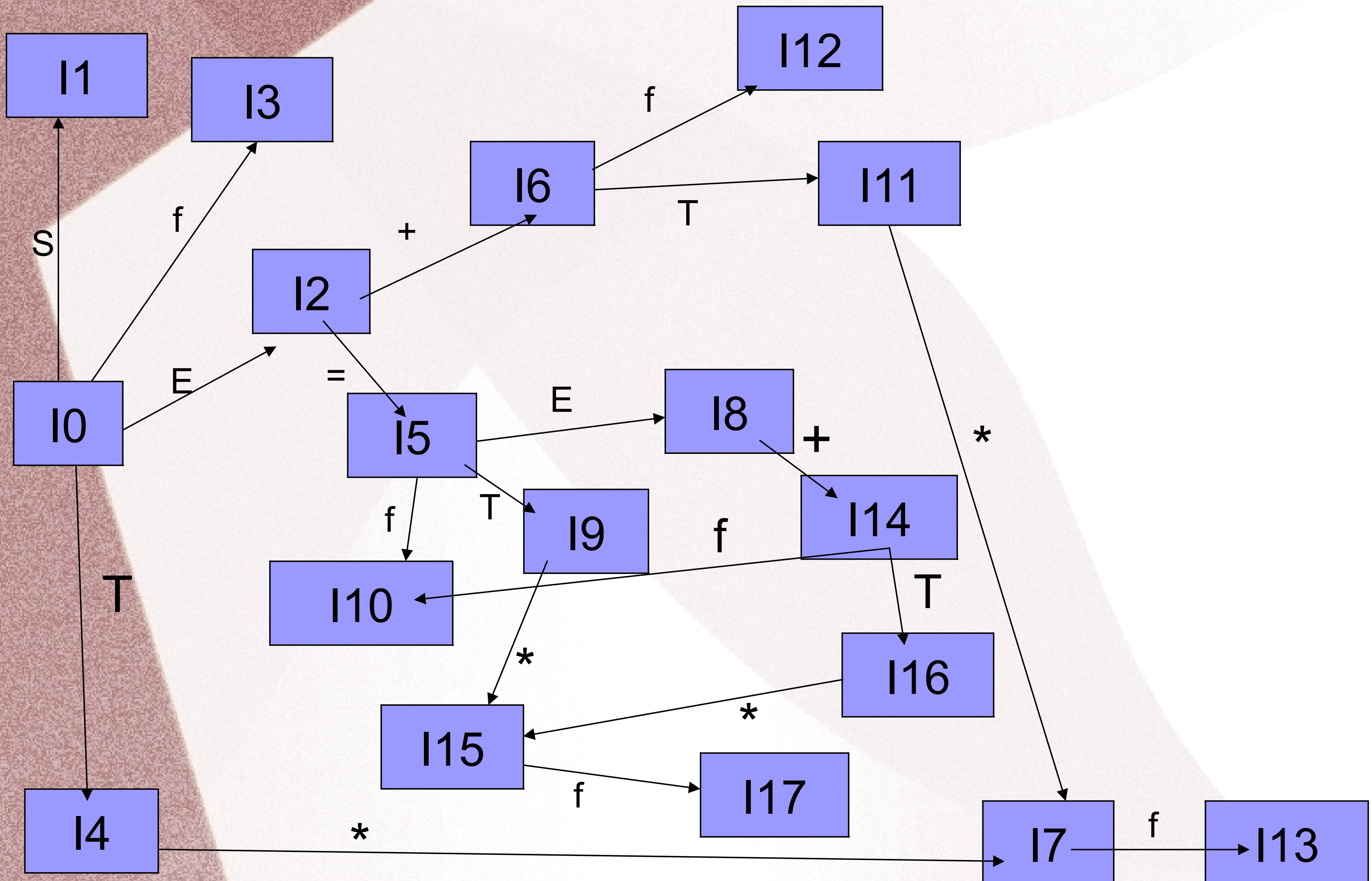
<u>State16 (I16) :</u>
<u>From state 14 on T</u>

E→E+T., $ +
T→T.*f, $ + *

<u>State17 (I17) :</u>
<u>From state 15 on f</u>

T→T*f. , $ + *

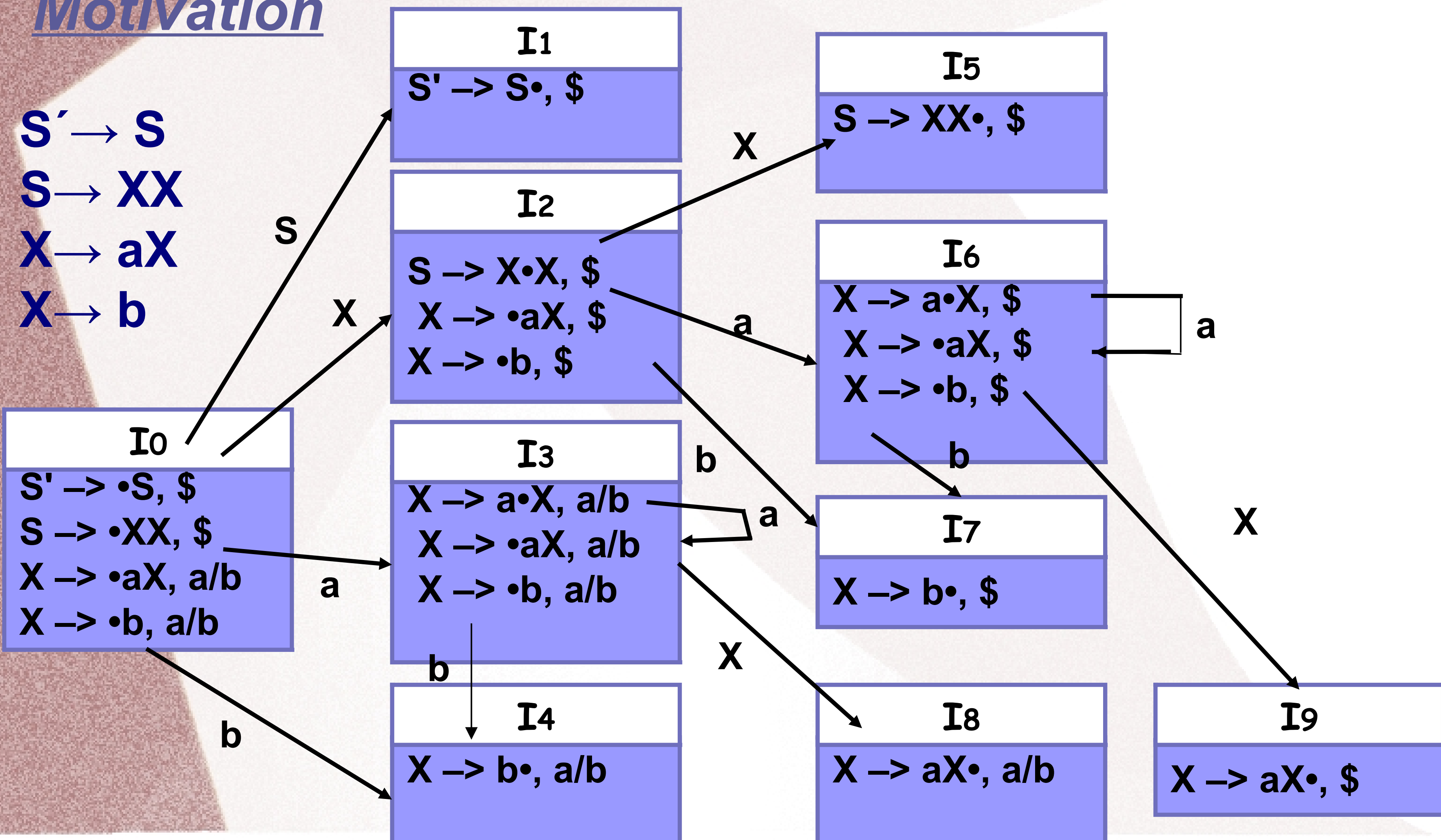
Finite Control of LR(1) parser



LALR parsing

Motivation

S´→ S
S→ XX
X→ aX
X→ b



LALR parsing

After Merging:

I_{36}: X -> a·X, a/b/$
X -> ·aX, a/b/$
X -> ·b, a/b/$

I_{47}: X -> b·, a/b/$

I_{89}: X -> aX·, a/b/$

LALR parsing

Example:

S' -> S
S -> Bbb | aab | bBa
B -> a

I0: S' -> •S, $
S -> •Bbb, $
S -> •aab, $
S -> •bBa, $
B -> •a, b

I1: S' -> S•, $

I2: S -> B•bb, $

I3: S -> a•ab, $
B -> a•, b
....

LALR merge conflict

Shift-Reduce conflict:

1. If LR (1) has shift-reduce conflict then LALR will also have it.
2. If LR (1) does not have shift-reduce conflict LALR will also not have it.
3. Any shift-reduce conflict which can be removed by LR (1) can also be removed by LALR.
4. If SLR has shift-reduce conflict then LALR may or may not remove it.
5. SLR and LALR tables for a grammar always have same number of states.

Hence, LALR parsing is the most suitable for parsing general programming languages. The table size is quite small as compared to LR (1) , and by carefully designing the grammar it can be made free of conflicts.

LALR merge conflict

Reduce-Reduce conflict: Example

S' → S
S → aBc
S → bCc
S → aCd
S → bBd
B → e
C → e

I0: S' -> •S, $
S -> •aBc, $
S -> •bCc, $
S -> •aCd, $
S -> •bBd, $

I1: S' -> S•, $

I2: S -> a•Bc, $
S -> a•Cd, $
B -> •e, c

I3: S -> b•Cc, $
S -> b•Bd, $
C -> •e, c
B -> •e, d

I4: S -> aB•c, $

I5: S -> aC•d, $

I6: B -> e•, c
C -> e•, d

I7: S -> bC•c, $

I8: S -> bB•d, $

I9: B -> e•, d
C -> e•, c

I10: S -> aBc•, $

I11: S -> aCd•, $

I12: S -> bCc•, $

I13: S -> bBd•, $