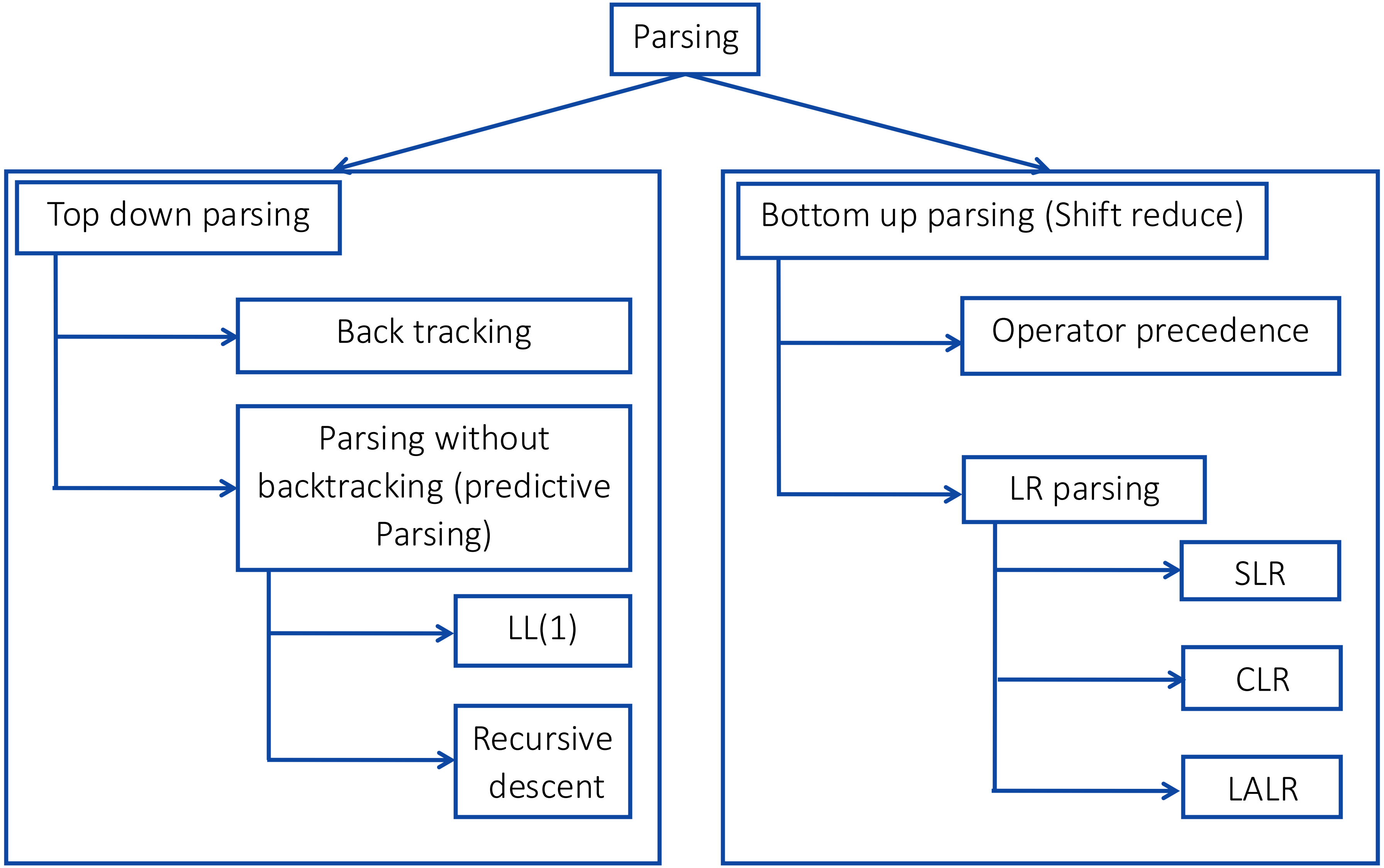
Module 2 – Syntax Analysis
Bottom Up Parsing

Parsing Methods



Handle & Handle pruning

- **Handle**: A "handle" of a string is a substring of the string that matches the right side of a production, and whose reduction to the non terminal of the production is one step along the reverse of rightmost derivation.
- **Handle pruning:** The process of discovering a handle and reducing it to appropriate left hand side non terminal is known as handle pruning.

E→E+E
E→E*E String: id1+id2*id3
E→id

Rightmost Derivation

E
E+E
E+E*E
E+E*id3
E+id2*id3
id1+id2*id3

Right sentential form	Handle	Production
id1+id2*id3		

Shift reduce parser

- The shift reduce parser performs following basic operations:

1. **Shift**: Moving of the symbols from input buffer onto the stack, this action is called shift.
2. **Reduce**: If handle appears on the top of the stack then reduction of it by appropriate rule is done. This action is called reduce action.
3. **Accept**: If stack contains start symbol only and input buffer is empty at the same time then that action is called accept.
4. **Error**: A situation in which parser cannot either shift or reduce the symbols, it cannot even perform accept action then it is called error action.

Example: Shift reduce parser

Grammar:

E→E+T | T

T→T*F | F

F→id

String: id+id*id

Stack	Input Buffer	Action

Consider the following grammar-

$S \rightarrow T\ L\ ;$
$T \rightarrow int \mid float$
$L \rightarrow L\ ,\ id \mid id$

Parse the input string int id , id ; using a shift-reduce parser.

Operator precedence parsing

- **Operator Grammar**: A Grammar in which there is no Є in RHS of any production or no adjacent non terminals is called operator grammar.
- Example: E→ EAE | (E) | id

  A→ + | * | -

- Above grammar is not operator grammar because right side *EAE* has consecutive non terminals.
- In operator precedence parsing we define following disjoint relations:

Relation	Meaning
a<·b	a "yields precedence to" b
a=b	a "has the same precedence as" b
a·>b	a "takes precedence over" b

Precedence & associativity of operators

Operator	Precedence	Associative
↑	1	right
*, /	2	left
+, -	3	left

Steps of operator precedence parsing

1. Find Leading and trailing of non terminal
2. Establish relation
3. Creation of table
4. Parse the string

Leading & Trailing

Leading:- Leading of a non terminal is the first terminal or operator in production of that non terminal.

Trailing:- Trailing of a non terminal is the last terminal or operator in production of that non terminal.

Example: $E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow id$

Non terminal	Leading	Trailing
E		
T		
F		

Rules to establish a relation

1. For a = b, $\Rightarrow \; aAb$, where A is ϵ or a single non terminal [e.g : (E)]
2. a <· b $\Rightarrow Op \,.\, NT \;\; then \;\; Op \;\; <. \, Leading(NT)$ [e.g : +T]
3. a ·> b $\Rightarrow NT \,.\, Op \; then \; (Trailing(NT)) \cdot > Op$ [e.g : E+]
4. $ <· Leading (start symbol)
5. Trailing (start symbol) ·> $

Example: Operator precedence parsing

Step 1: Find Leading & Trailing of NT

Nonterminal	Leading	Trailing
E	{+,*,id}	{+,*,id}
T	{*,id}	{*,id}
F	{id}	{id}

E→ E+T | T
T→ T *F| F
F→ id

Step 2: Establish Relation

a <·b

$$Op \cdot NT \quad | \; Op \lessdot Leading(NT)$$

$$+T \quad | \; + \lessdot \{*, id\}$$

$$* F \quad | \; * \lessdot \{id\}$$

Step3: Creation of Table

	+	*	id	$
+				
*				
id				
$				

Example: Operator precedence parsing

Step 1: Find Leading & Trailing of NT

Nonterminal	Leading	Trailing
E	{+,*,id}	{+,*,id}
T	{*,id}	{*,id}
F	{id}	{id}

E→ E+ T| T
T→ T* F| F
F→ id

Step2: Establish Relation

a ·>b

$$NT \cdot Op \quad (Trailing(NT)) \cdot > Op$$

$$E + \quad \{+,*,id\} \cdot > \; +$$

$$T * \quad \{*,id\} \cdot > *$$

Step3: Creation of Table

	+	*	id	$
+		<·	<·	
*			<·	
id				
$				

Example: Operator precedence parsing

Step 1: Find Leading & Trailing of NT

Nonterminal	Leading	Trailing
E	{+,*,id}	{+,*,id}
T	{*,id}	{*,id}
F	{id}	{id}

E→ E+ T| T
T→ T* F| F
F→ id

Step 2: Establish Relation

$<· Leading (start symbol)

$\$ \lessdot \{+, *, id\}$

Trailing (start symbol) ·> $

$\{+, *, id\} \gtrdot \ \$$

Step 3: Creation of Table

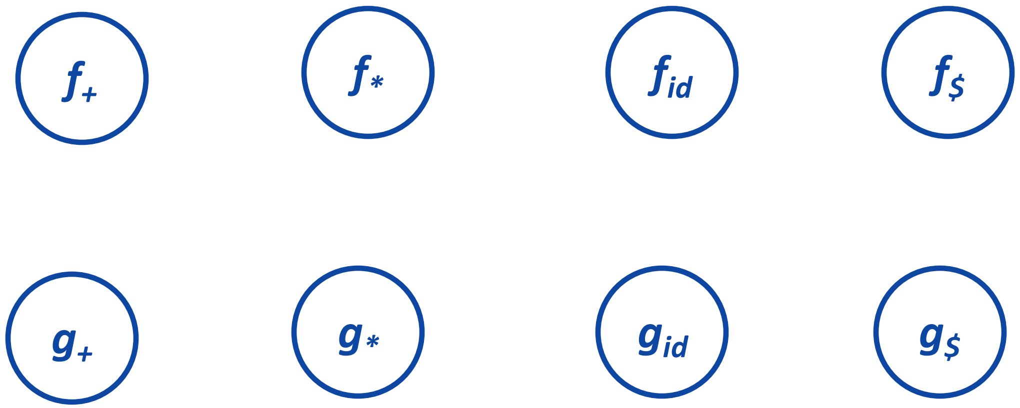
	+	*	id	$
+	·>	<·	<·	
*	·>	·>	<·	
id	·>	·>		
$				

Operator precedence function

1. Create functions f_a and g_a for each a that is terminal or $.

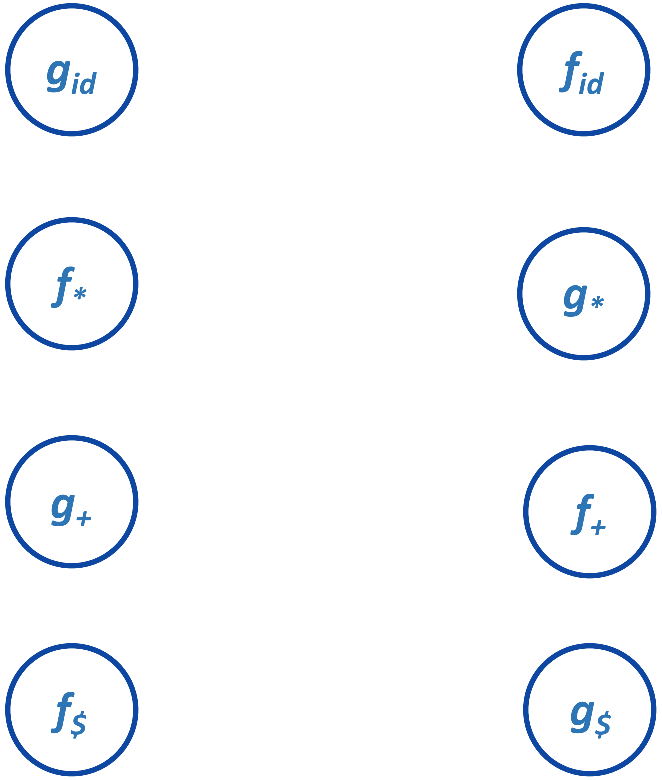
E→ E+T | T
T→ T*F | F
F→ id

$$a = \{+, *, id\}\ or\ \$$$



Operator precedence function

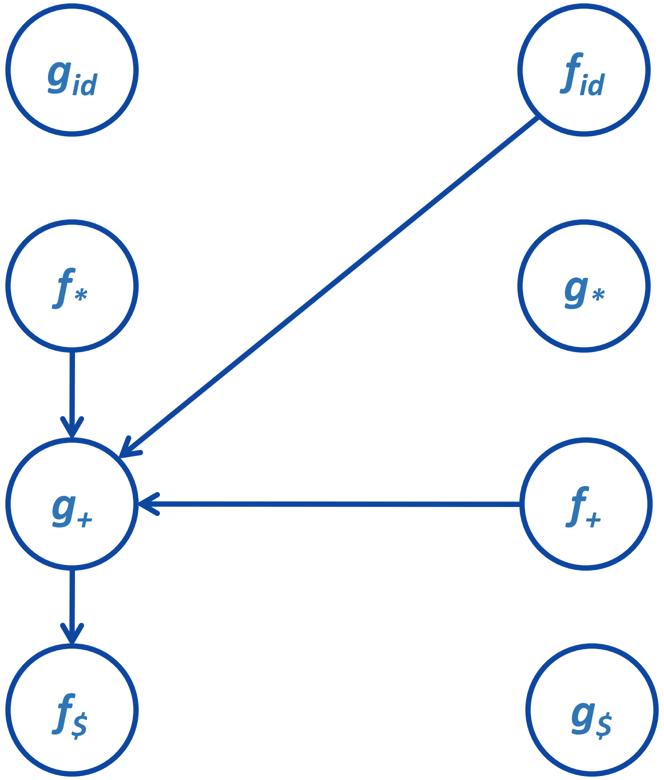
- *Partition the symbols in as many as groups possible, in such a way that f_a and g_b are in the same group if $a \doteq b$.*



	+	*	id	$
+	·>	<·	<·	·>
*	·>	·>	<·	·>
id	·>	·>		·>
$	<·	<·	<·	

Operator precedence function

3. *if a <· b, place an edge from the group of g_b to the group of f_a*
   if a ·> b, place an edge from the group of f_a to the group of g_b



		g			
f		**+**	*****	**id**	**$**
	+	·>	<·	<·	·>
	*****	·>	·>	<·	·>
	id	·>	·>		·>
	$	<·	<·	<·	

$f_+ \cdot\!> g_+$ $\quad f_+ \rightarrow g_+$

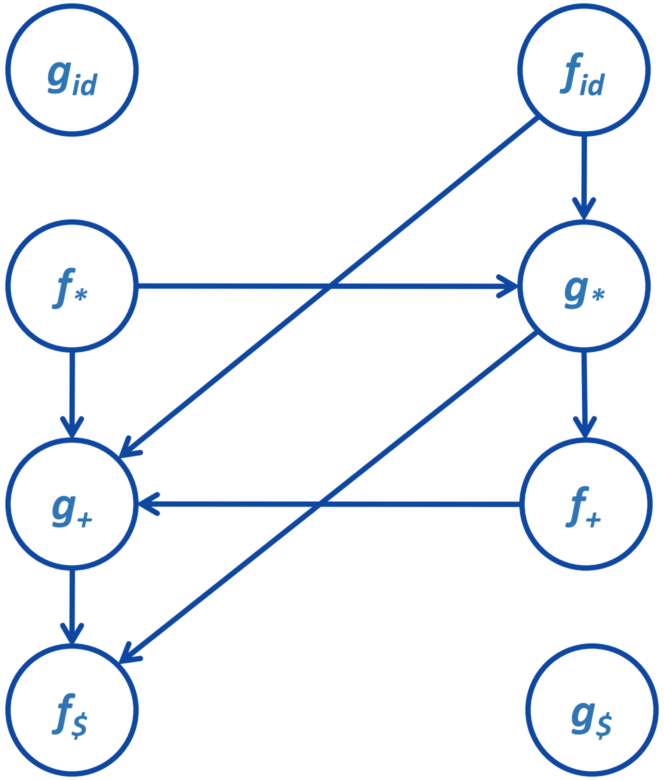
$f_* \cdot\!> g_+$ $\quad f_* \rightarrow g_+$

$f_{id} \cdot\!> g_+$ $\quad f_{id} \rightarrow g_+$

$f_\$ <\!\cdot\ g_+$ $\quad f_\$ \leftarrow g_+$

Operator precedence function

3. *if a <· b, place an edge from the group of g_b to the group of f_a*
   if a ·> b, place an edge from the group of f_a to the group of g_b

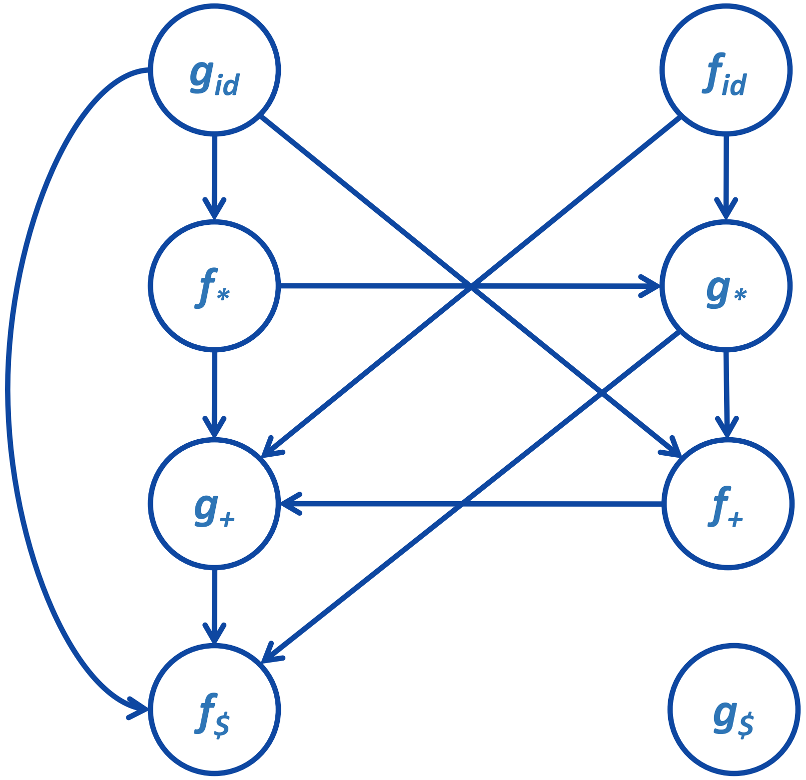


f \ g	+	*	id	$
+	·>	<·	<·	·>
*	·>	·>	<·	·>
id	·>	·>		·>
$	<·	<·	<·	

$f_+ <\cdot g_*$	$f_+ \leftarrow g_*$
$f_* \cdot> g_*$	$f_* \rightarrow g_*$
$f_{id} \cdot> g_*$	$f_{id} \rightarrow g_*$
$f_\$ <\cdot g_*$	$f_\$ \leftarrow g_*$

Operator precedence function

3. *if a <· b, place an edge from the group of g_b to the group of f_a*
   if a ·> b, place an edge from the group of f_a to the group of g_b



		g			
f		**+**	*****	**id**	**$**
	+	·>	<·	<·	·>
	*****	·>	·>	<·	·>
	id	·>	·>		·>
	$	<·	<·	<·	

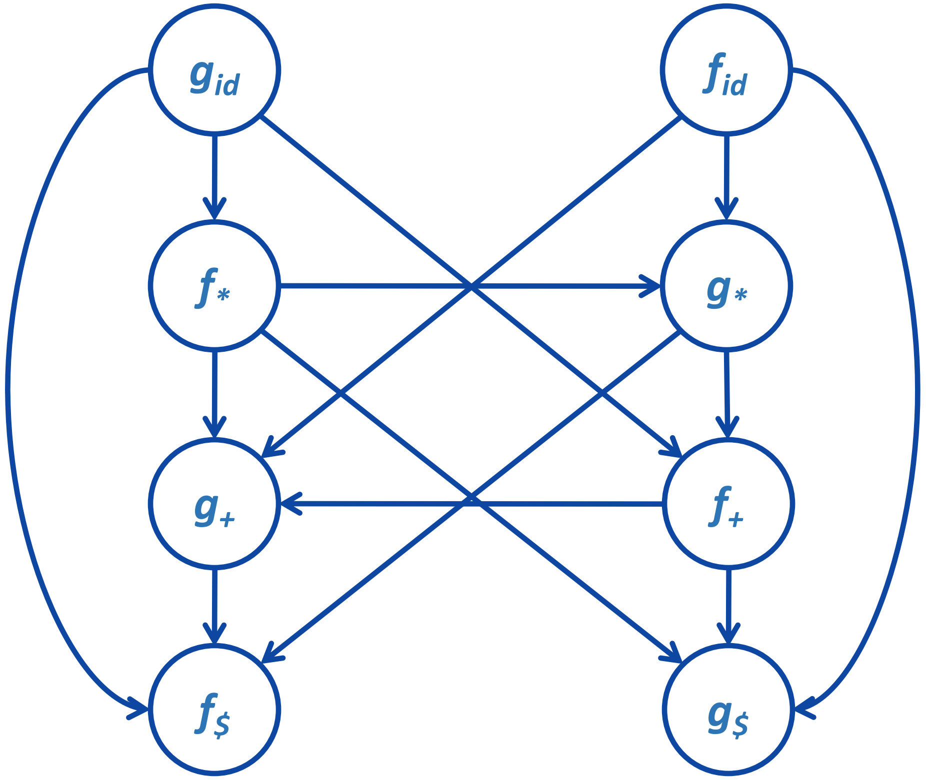
$f_+ <\cdot\, g_{id}$ $\qquad f_+ \leftarrow g_{id}$

$f_* <\cdot\, g_{id}$ $\qquad f_* \leftarrow g_{id}$

$f_\$ <\cdot\, g_{id}$ $\qquad f_\$ \leftarrow g_{id}$

Operator precedence function

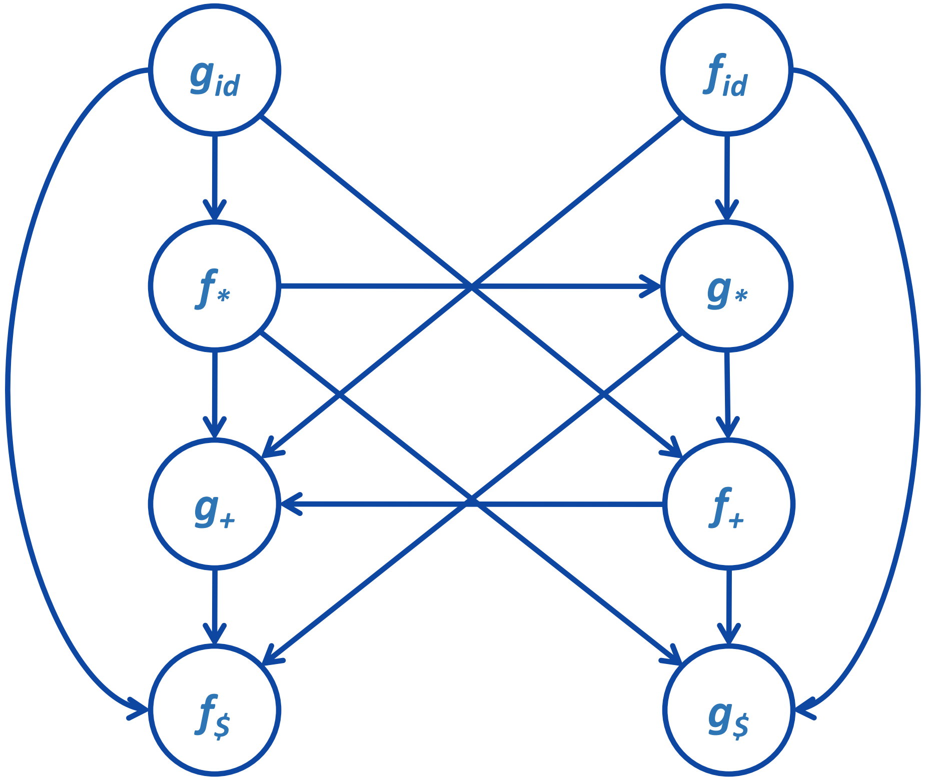
3. *if a <· b, place an edge from the group of g_b to the group of f_a*
if a ·> b, place an edge from the group of f_a to the group of g_b



		g			
f		+	*	id	$
	+	·>	<·	<·	·>
	*	·>	·>	<·	·>
	id	·>	·>		·>
	$	<·	<·	<·	

$f_+ <\cdot g_\$$	$f_+ \rightarrow g_\$$
$f_* <\cdot g_\$$	$f_* \rightarrow g_\$$
$f_{id} <\cdot g_\$$	$f_{id} \rightarrow g_\$$

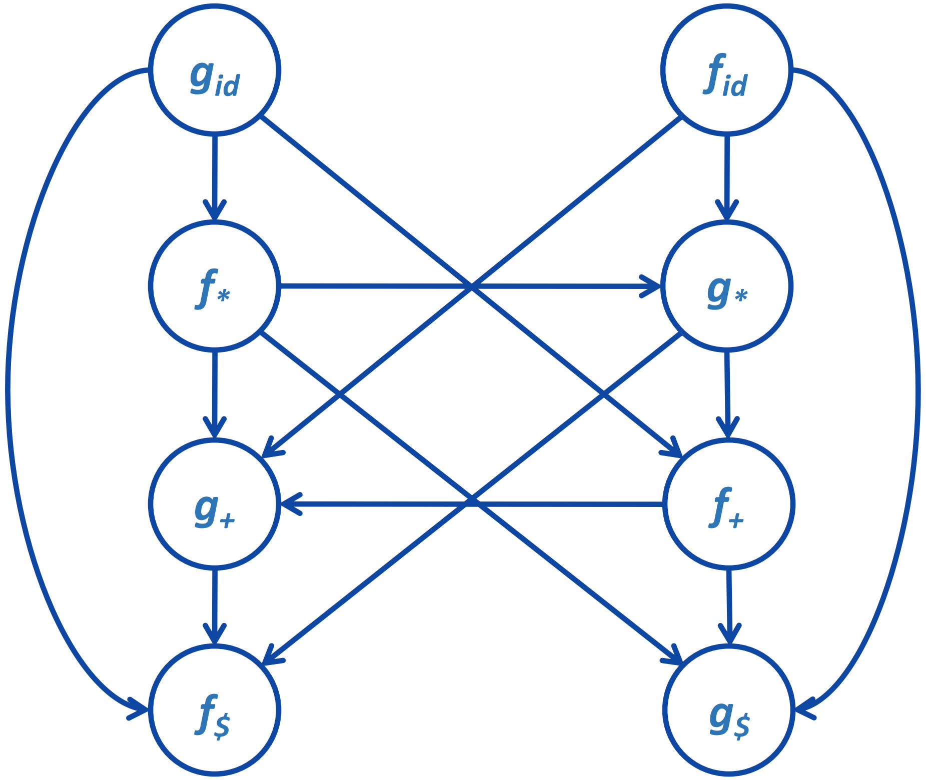
Operator precedence function



	+	*	id	$
f				
g				

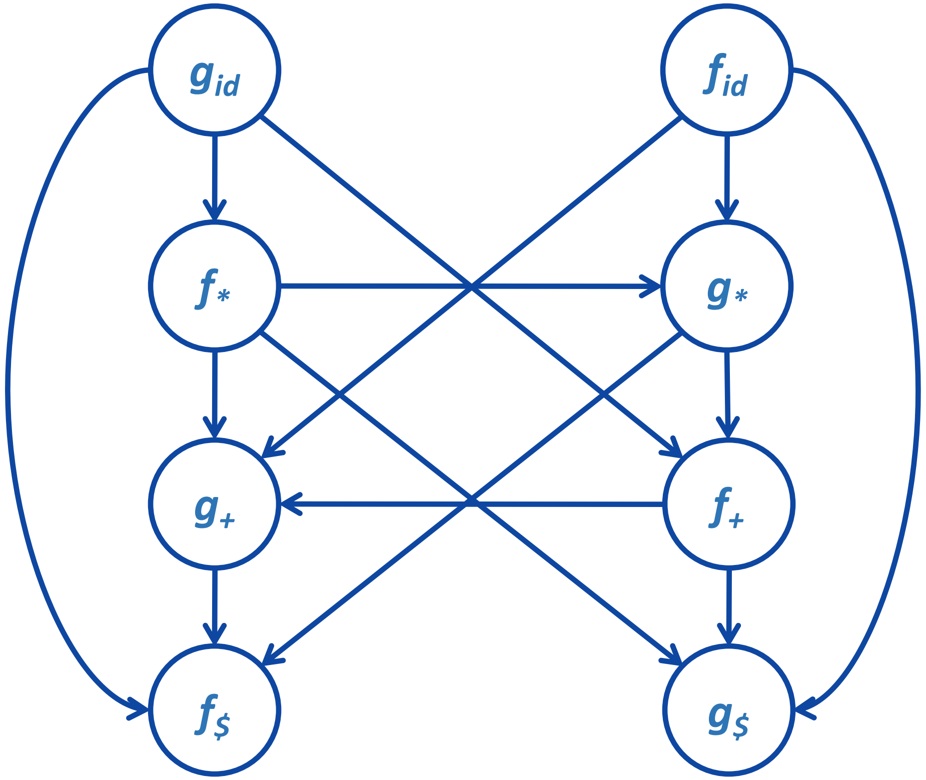
4. *If the constructed graph has a cycle then no precedence functions exist. When* ~~*there are no cycles collect the length of the longest paths from the groups of*~~ $\cancel{f_a}$ ~~*and*~~ $\cancel{g_b}$ ~~*respectively.*~~

Operator precedence function



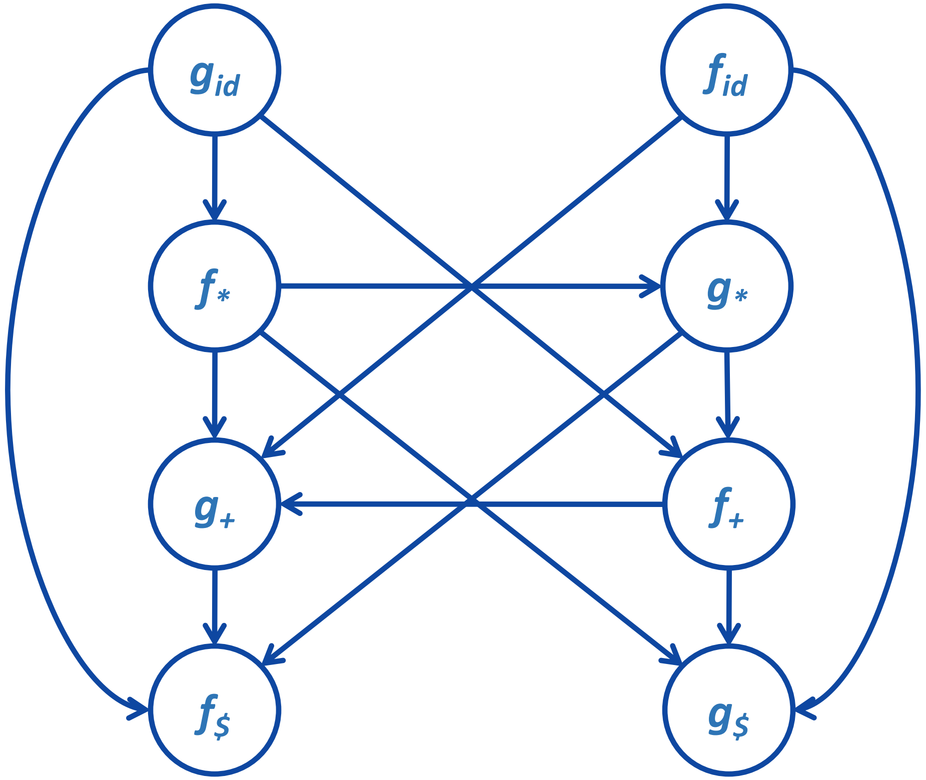
	+	*	id	$
f	2			
g				

Operator precedence function



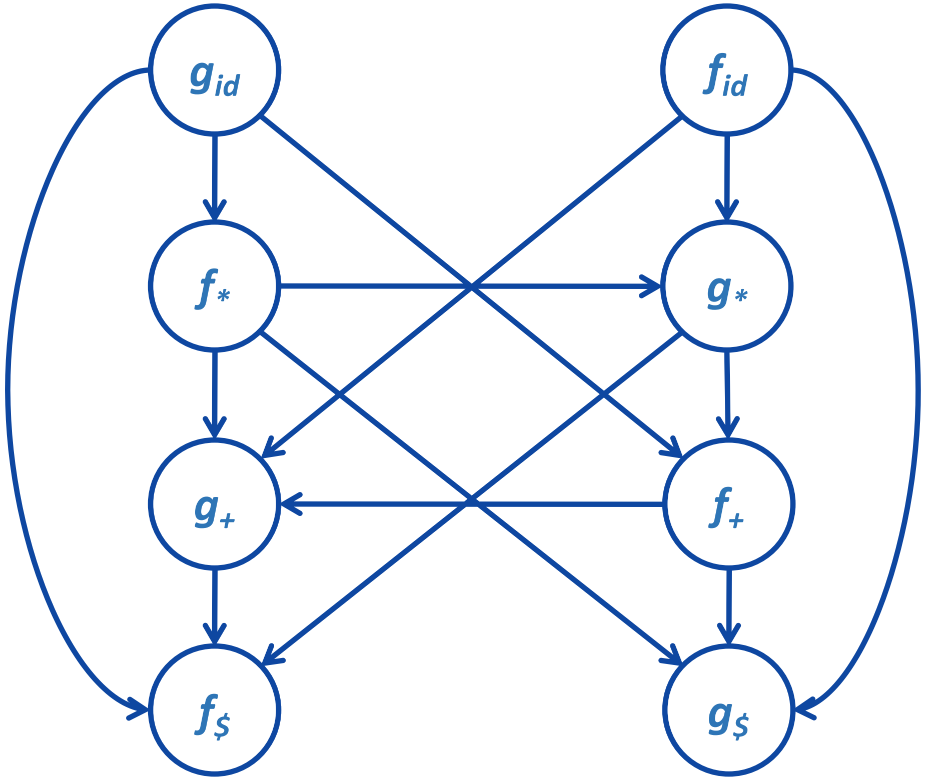
	+	*	id	$
f	2			
g	1			

Operator precedence function



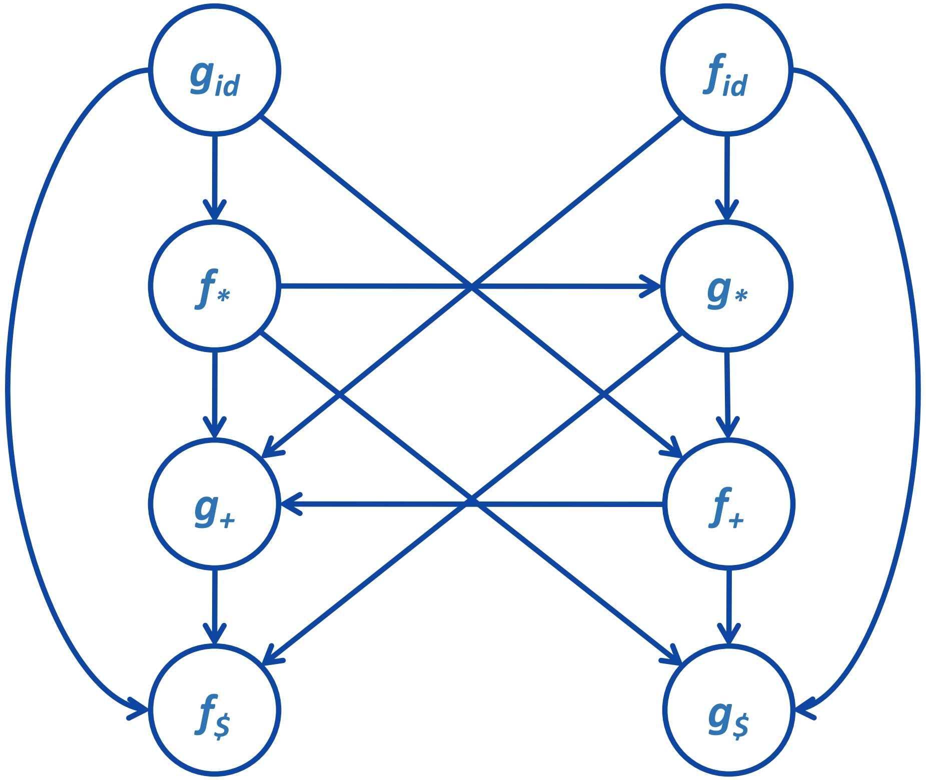
	+	*	id	$
f	2	4		
g	1			

Operator precedence function



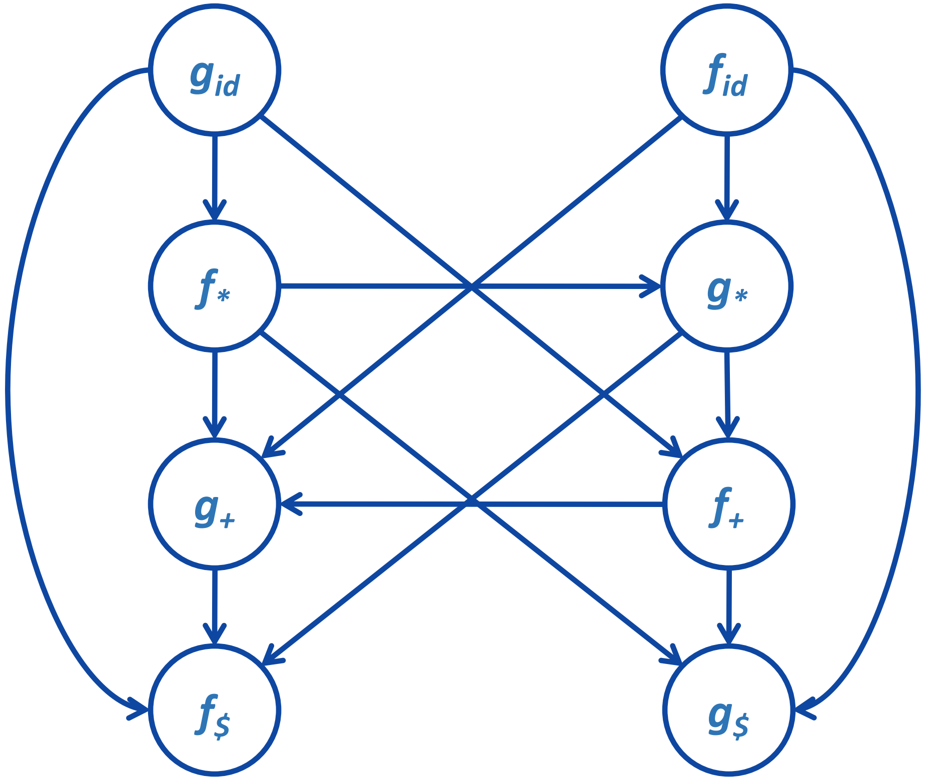
	+	*	id	$
f	2	4		
g	1	3		

Operator precedence function



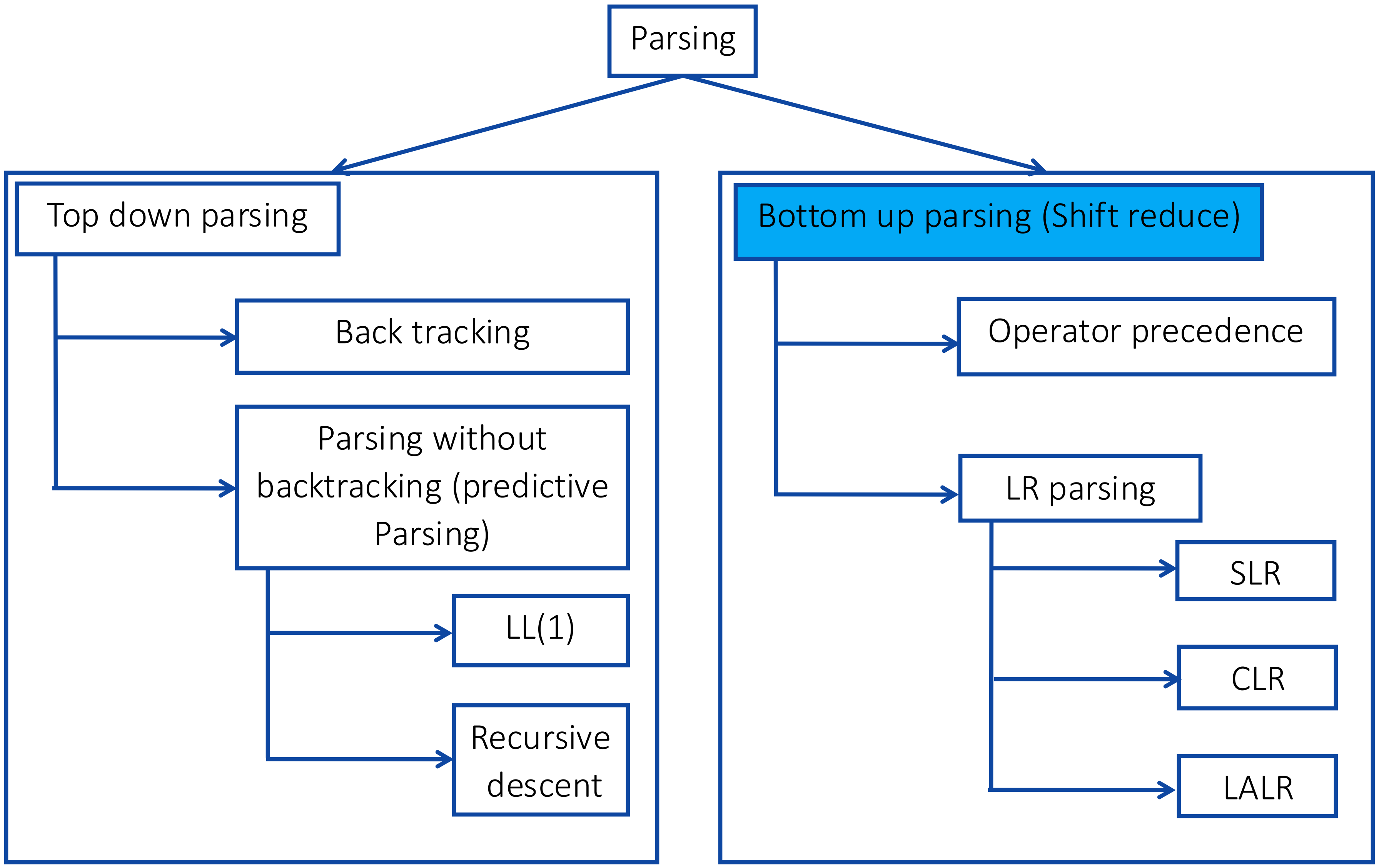
	+	*	id	$
f	2	4	4	
g	1	3		

Operator precedence function



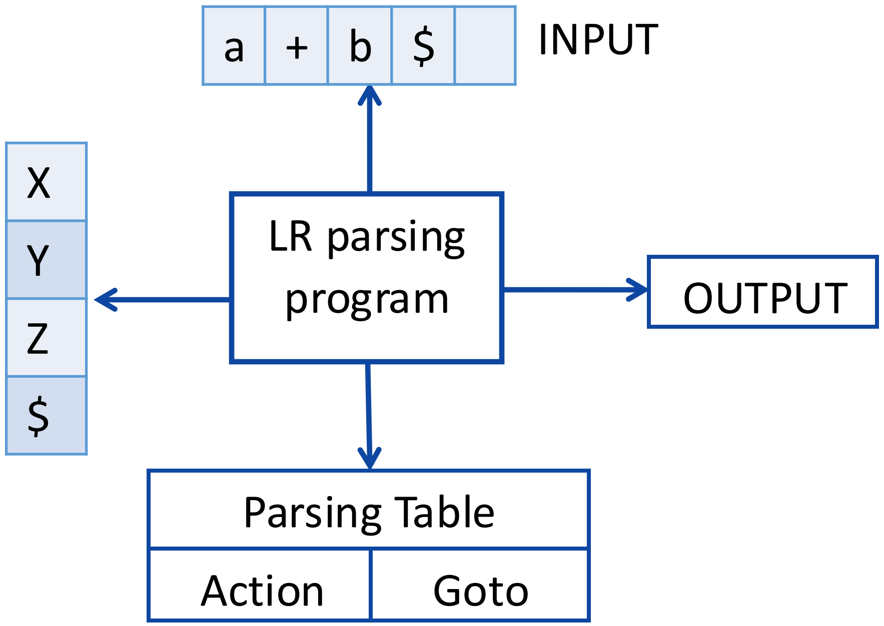
	+	*	id	$
f	2	4	4	
g	1	3	5	

Parsing Methods

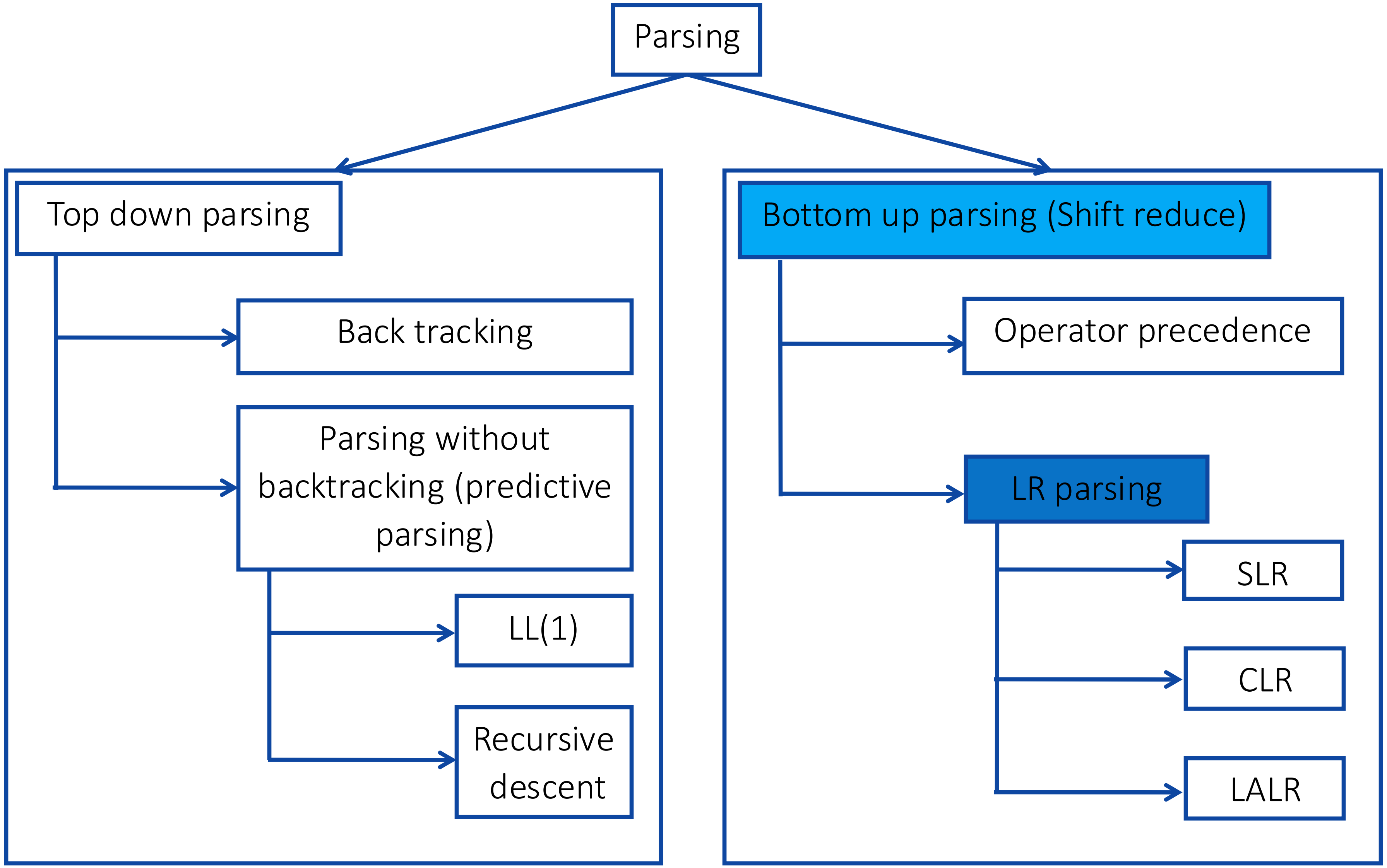


LR parser

- LR parsing is most efficient method of bottom up parsing which can be used to parse large class of context free grammar.
- The technique is called LR(k) parsing:
    1. The "L" is for left to right scanning of input symbol,
    2. The "R" for constructing right most derivation in reverse,
    3. The "k" for the number of input symbols of look ahead that are used in making parsing decision.



Parsing Methods



Computation of closure & go to function

$X \rightarrow Xb$

Closure(I):

$X \rightarrow .X\,b$

Goto(I,X)

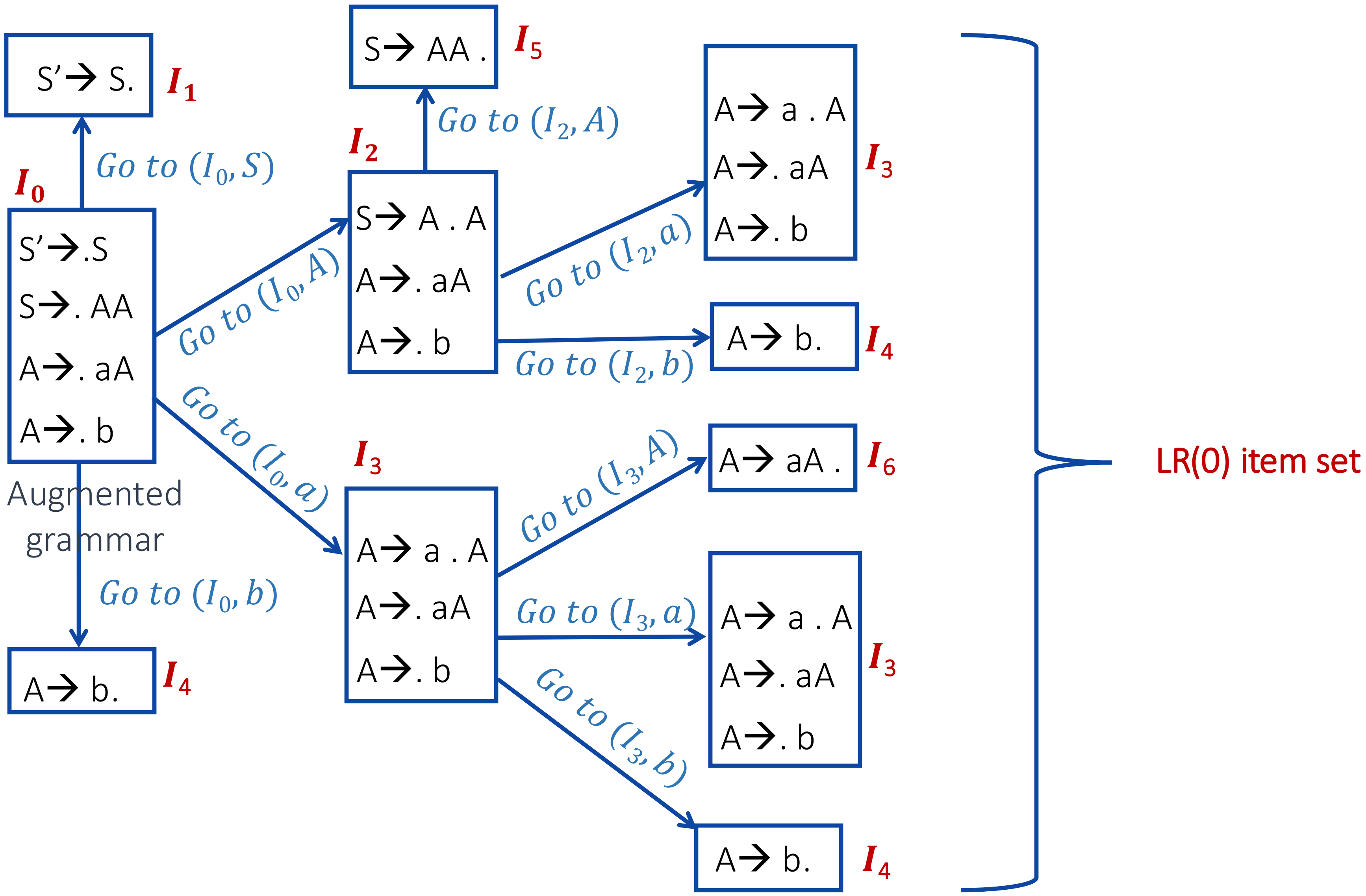
$X \rightarrow .X\,b$

Steps to construct SLR parser

1. Construct Canonical set of LR(0) items
2. Construct SLR parsing table
3. Parse the input string

Example: SLR(1)- simple LR

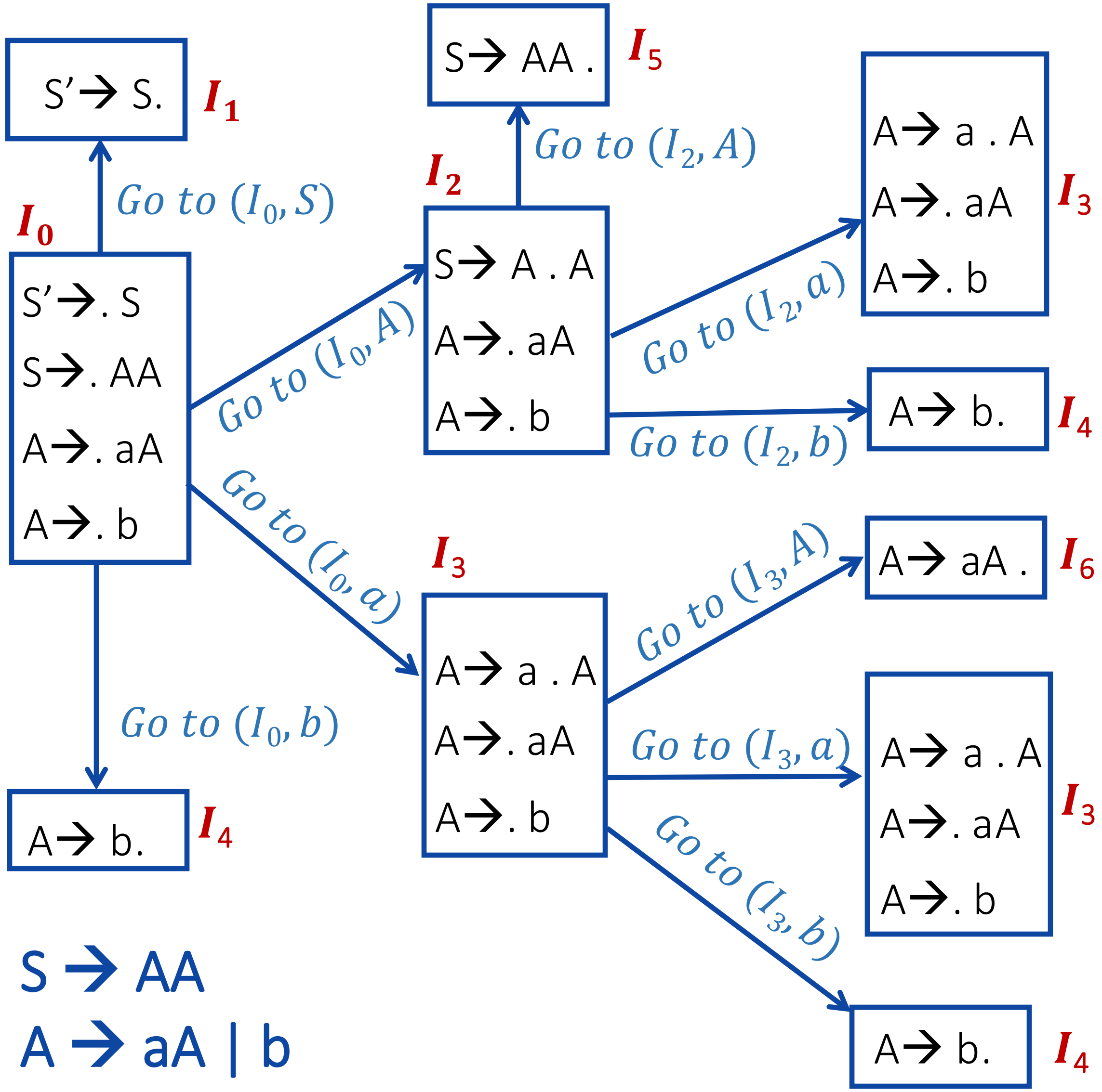
S → AA
A → aA | b



Rules to construct SLR parsing table

1. Construct $C = \{ I_0, I_1, \dots \dots . In\}$, the collection of sets of LR(0) items for G'.
2. State i is constructed from I_i. The parsing actions for state i are determined as follow :
   a) If $[A \rightarrow \alpha . a\beta]$ is in I_i and GOTO $(Ii , a) = I_j$, then set $ACTION[i, a]$ to "shift j". Here a must be terminal.
   b) If $[A \rightarrow \alpha .]$ is in I_i, then set $ACTION[i, a]$ to "reduce A→ α" for all a in $FOLLOW(A)$; here A may not be S'.
   c) If $[S \rightarrow S.]$ is in I_i, then set action $[i, \$]$ to "accept".
3. The goto transitions for state i are constructed for all non terminals A using the$if\ GOTO(Ii , A) = I_j\ then\ GOTO\ [i, A] = j$.
4. All entries not defined by rules 2 and 3 are made error.

Example: SLR(1)- simple LR



S → AA

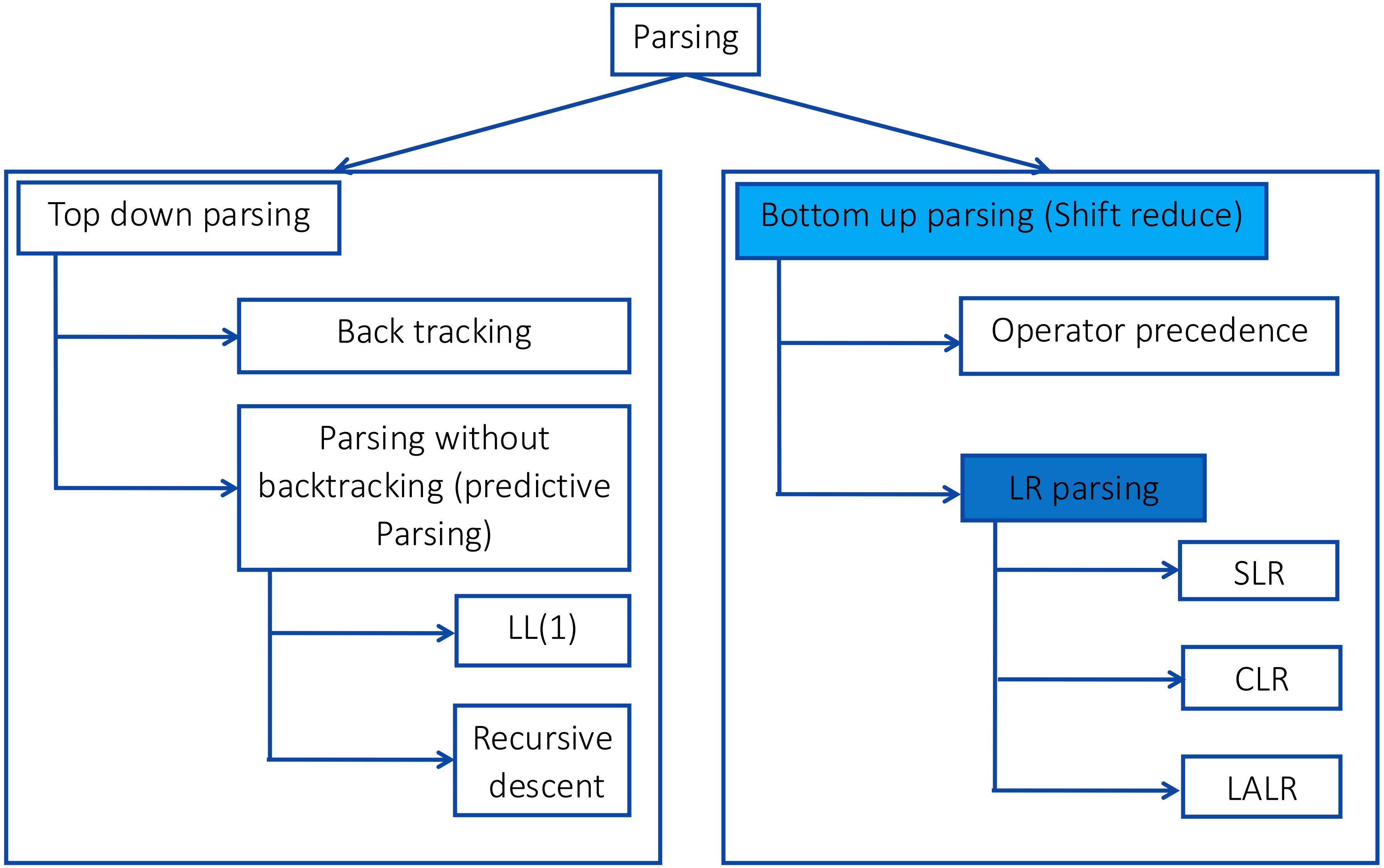
A → aA | b

$Follow(S) = \{\$\}$

$Follow(A) = \{a, b, \$\}$

	Action			Go to	
Item set	a	b	$	S	A
0					
1					
2					
3					
4					
5					
6					

Parsing Methods



How to calculate look ahead?

How to calculate look ahead?

S→CC

C→ cC | d

Closure(I)

S'→.S,$

S→.CC, $

C→.cC,c|d

C→.d, c|d

S'	→		.	S		,	$
A	→	α	.	X	β	,	a

$\text{Lookahead} = \text{First}(\boldsymbol{\beta a})$

First($)

$= \$$

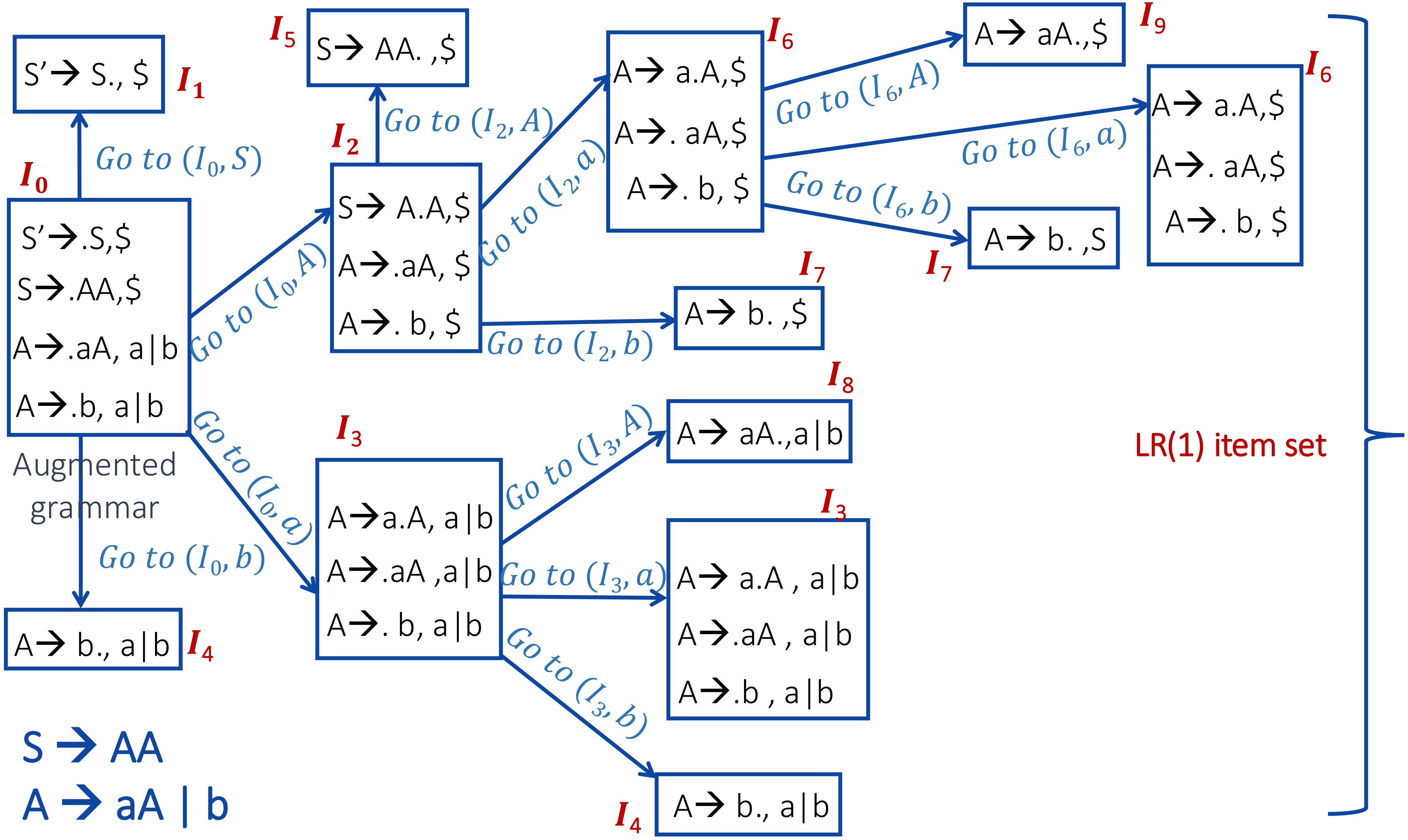
S	→		.	C	C	,	$
A	→	α	.	X	β	,	a

$\text{Lookahead} = \text{First}(\boldsymbol{\beta a})$

$\text{First}(C\$)$

$= \boldsymbol{c, d}$

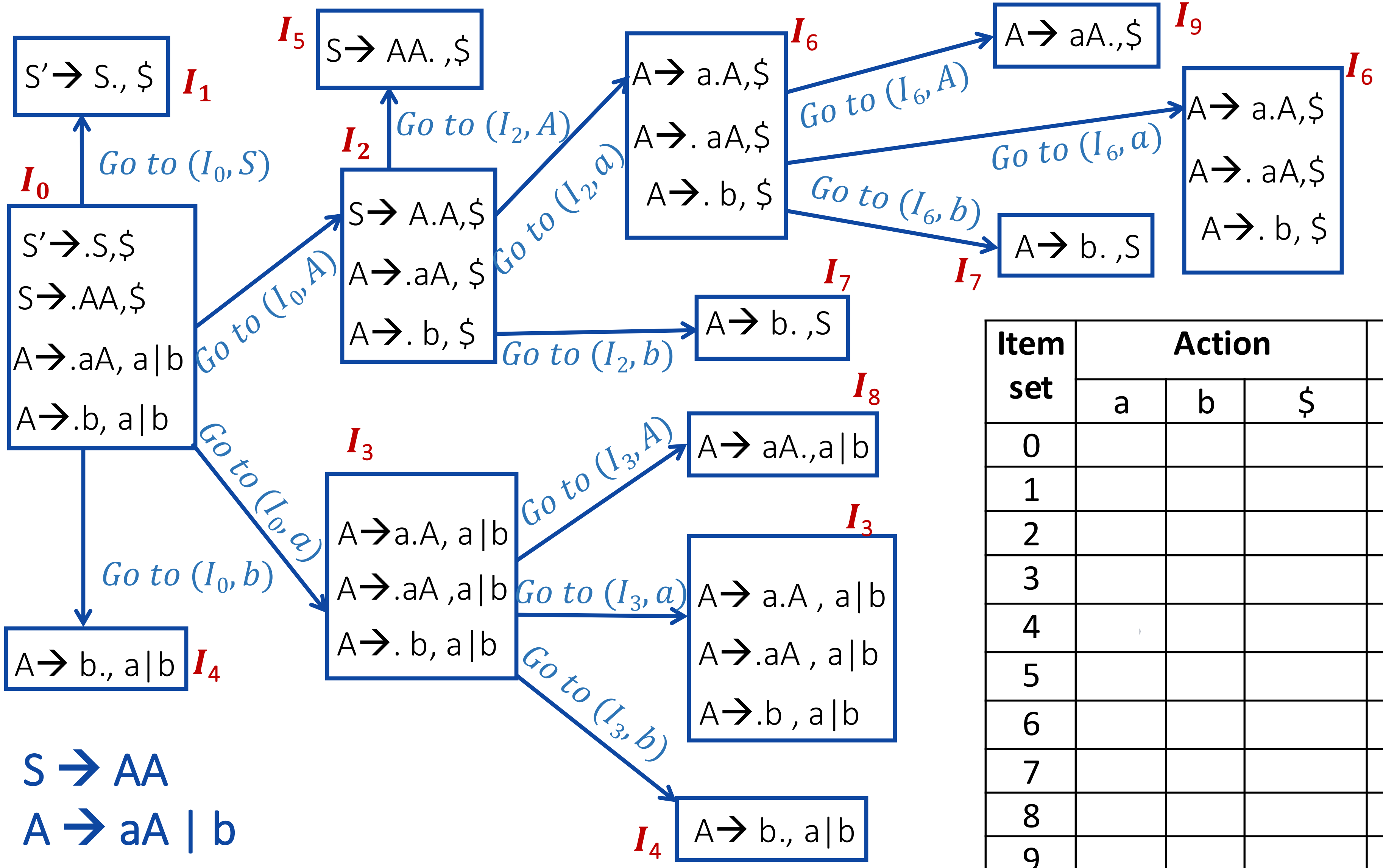
Example: CLR(1)- canonical LR



$S \rightarrow AA$

$A \rightarrow aA \mid b$

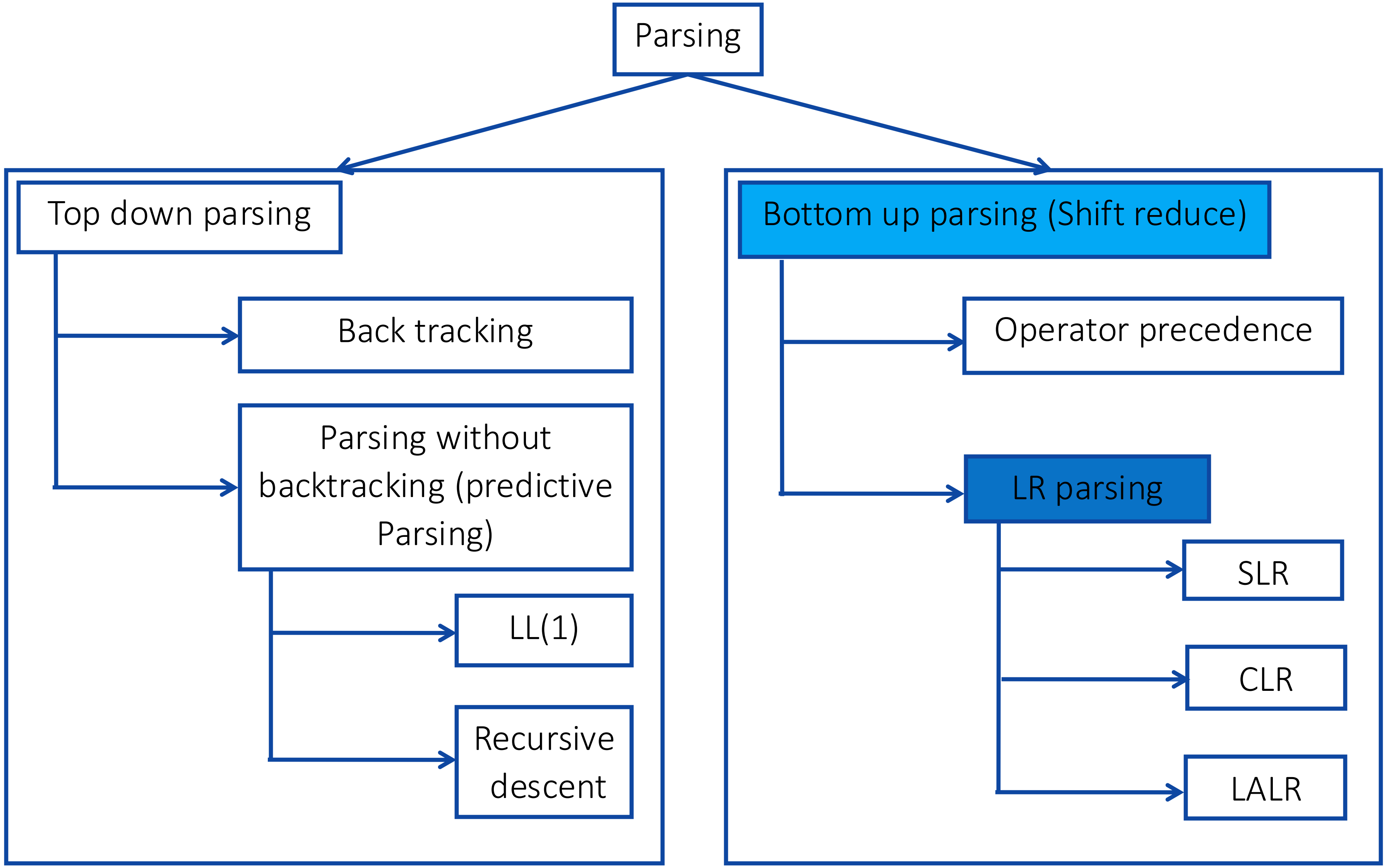
Example: CLR(1)- canonical LR



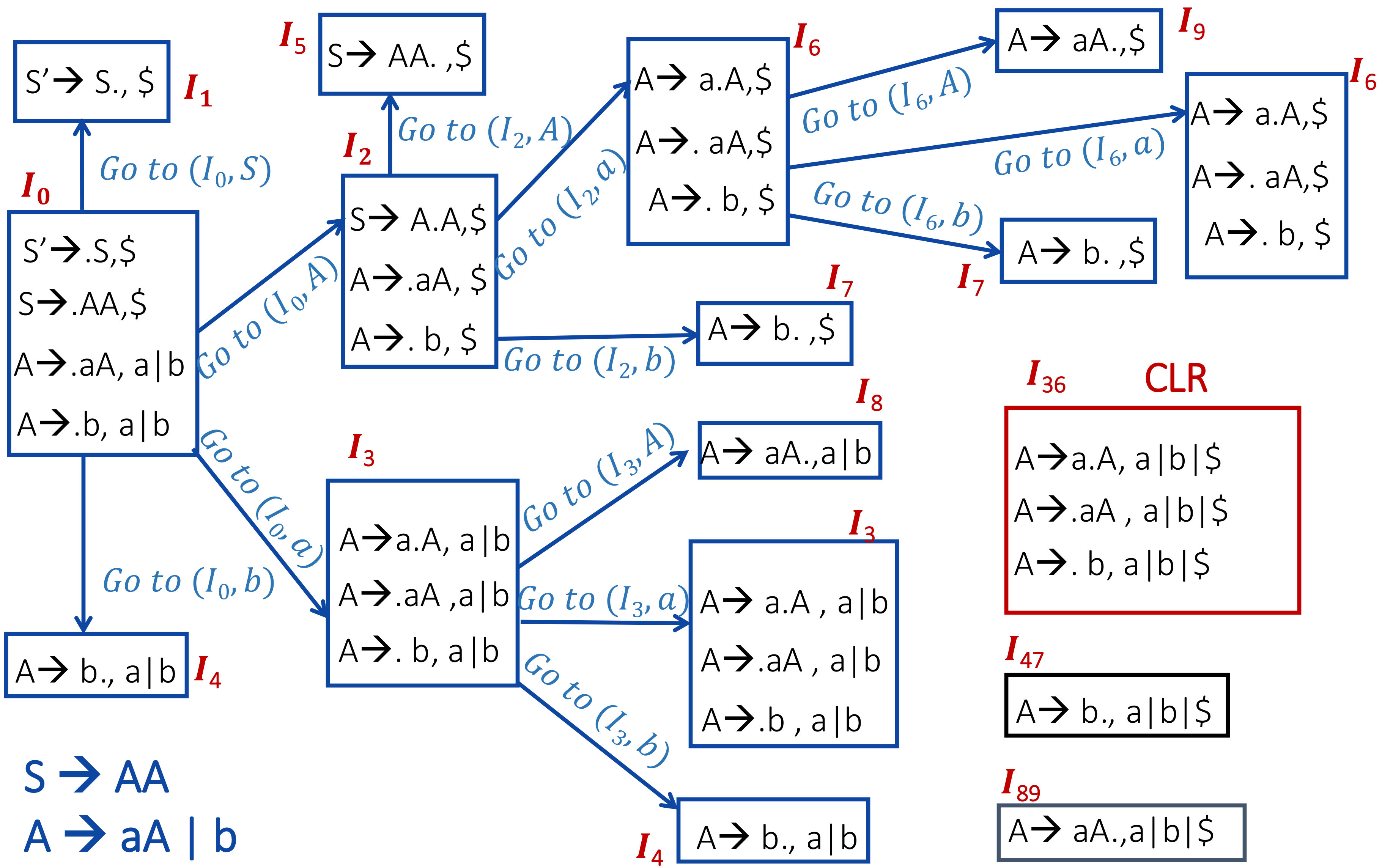
S → AA
A → aA | b

Item set	Action			Go to	
	a	b	$	S	A
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					

Parsing Methods



Example: LALR(1)- look ahead LR



Example: LALR(1)- look ahead LR

Item set	Action			Go to	
	a	b	$	S	A
0	S3	S4		1	2
1			Accept		
2	S6	S7			5
3	S3	S4			8
4	R3	R3			
5			R1		
~~6~~	~~S6~~	~~S7~~			~~9~~
~~7~~			~~R3~~		
8	R2	R2			
~~9~~			~~R2~~		

CLR Parsing Table

Item set	Action			Go to	
	a	b	$	S	A
0	S36	S47		1	2
1			Accept		
2	S36	S47			5
36	S36	S47			89
47	R3	R3	R3		
5			R1		
89	R2	R2	R2		

LALR Parsing Table