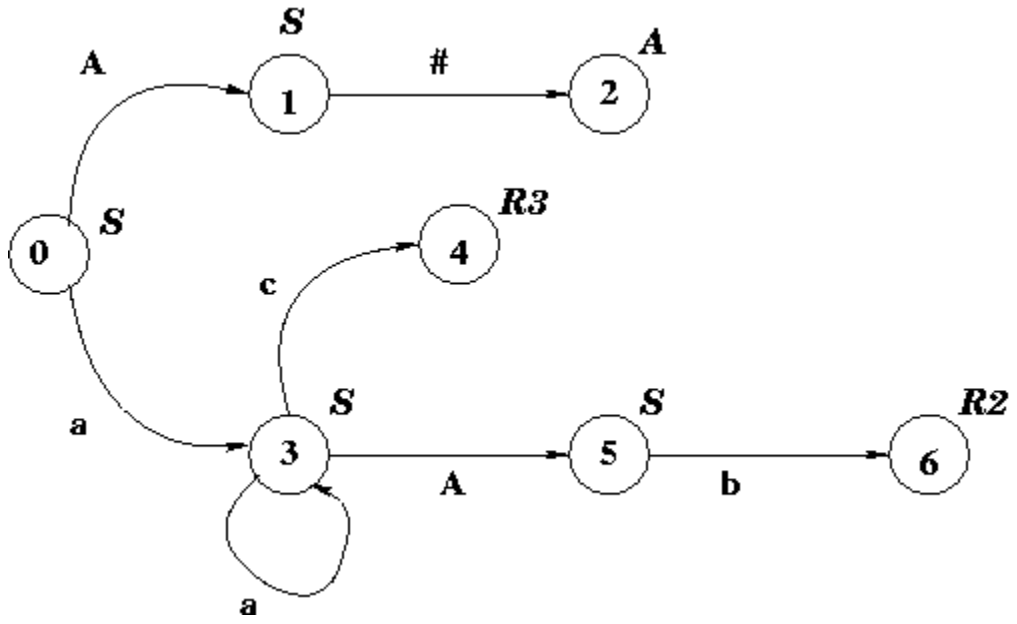
LR(0)

Let's consider the following grammar

1. S → A #
2. A → a A b
3. | c

An LR(0) FSA for the above grammar is



In the above LR(0) FSA states 1, 3, and 5 are stack states; state 0 is the initial state; states 4 and 6 are reduce states; and state 2 is the accept state.

Def: An *LR parse configuration* is a 5-tuple (N, S, α, A, π) where N is the next state, S is the stack, α is the input, A is the action(Stack|Reduce), and π is the reduction).

LR(0) Parse Algorithm

1. Initialize the stack with the start state.
2. Read an input symbol
3. while true do
   - 3.1 Using the top of the stack and the input symbol determine the next state.
   - 3.2 If the next state is a stack state

     then

3.2.1 stack the state
3.2.2 get the next input symbol

3.3 else if the next state is a reduce state

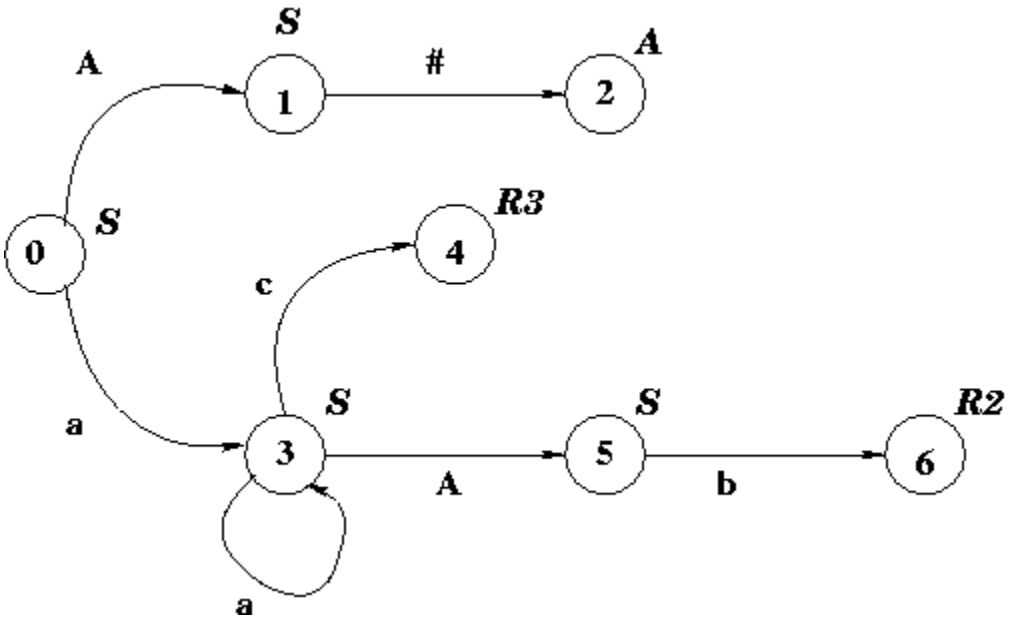
then
3.3.1 output reduction number, k
3.3.2 pop $|RHS_k|$-1 states from the stack where RHS_k is the right hand side of production k.
3.3.3 set the next input symbol to the LHS_k

3.4 else if the next state is an accept state

then
3.4.1 output valid sentence
3.4.2 return

else
3.4.3 output invalid sentence
3.4.4 return

Let's parse the sentence a a c b b # using the LR(0) FSA below:



Input Symbol	N	S	α	A/π	Grammar
	-	0	a a c b b #	-	1. S → A # 2. A → a A b 3. | c

Click here for the complete parse sequence.

Now let's see how the above LR(0) FSA was obtained.

Def: An *item* or a *configuration* of a given grammar G is a marked production of the form

$$[\ A \rightarrow \alpha_1 \bullet \alpha_2\ ]$$

where $A \rightarrow \alpha_1\alpha_2 \in P$ and the period or dot denotes the mark.

Note: An item $[\ A \rightarrow \alpha_1 \bullet \alpha_2\ ]$ is valid for some viable prefix $\phi\alpha_1$ if and only if there exists some rightmost derivation

$$S \Rightarrow_R^* \phi A \tau \Rightarrow_R \phi \alpha_1 \alpha_2 \tau$$

in which τ is a string of terminal symbols.

Construct LR(0) Item Sets

1. Start with C_0 by including all marked productions **[S → • α]**

2. Compute the closure of the item set C_0 by
   If **[A → • X α_2]** where $X \in V_n$, we include in C_0 all items of the form **[X → • β].**

3. Perform a read operation on items in an item set,
   e.g. if **[A → α • X β]** is in the item set, then read X yielding a new item set whose initial member is **[A → α X • β]**.

4. Compute the closure of the new item set.
   If **[A → α_1 • X α_2]** where $X \in V_n$, we include in C_0 all items of the form **[X → • β].**

5. Continue reading until all •s have traveled through all item sets.

Note: Reduce states are associated with a completed item, i.e. where the • has traveled through an item set.

Repeating the above grammar

1. S → A #
2. A → a A b
3. | c

The LR(0) item sets for this grammar are

C_0

Click here for the LR(0) item sets.

As another example consider the following grammar:

1. S → E #
2. E → T
3. | E + T
4. | E - T
5. T → i
6. | (E)

The LR(0) item sets for the above grammar are

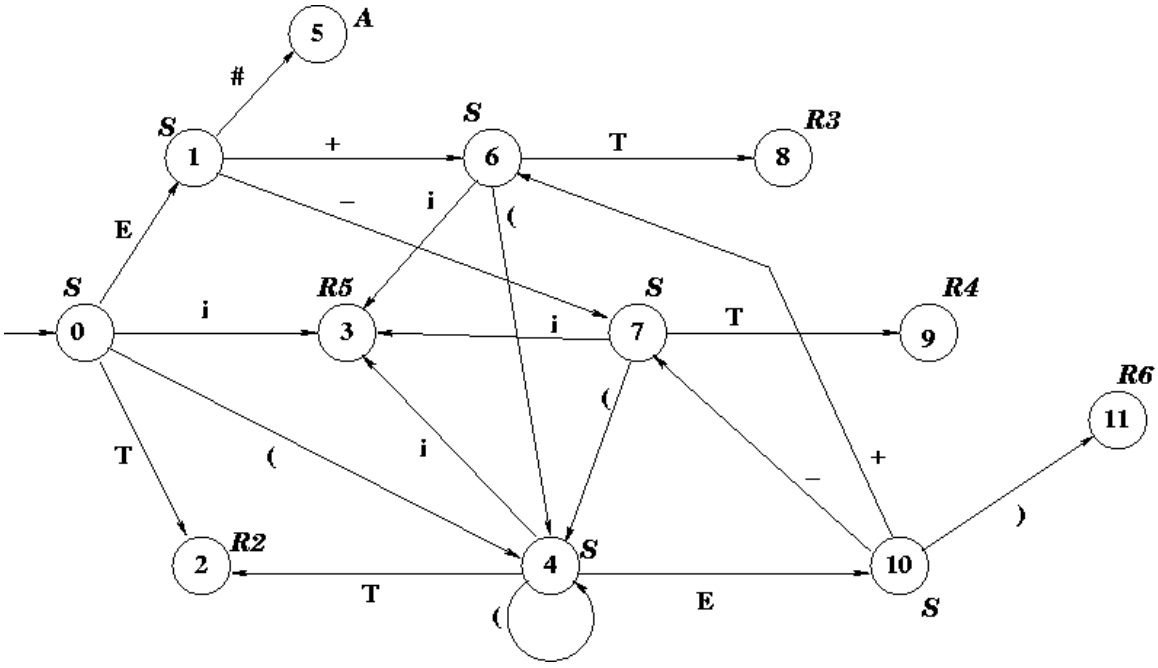
C_0

Click here for all of the LR(0) item sets.
Using the partial item sets for the grammar below

State	Basis Set	Closure Set	Next State or Reduce
0	[S → • E #]		1
		[E → • T]	2
		[E → • E + T]	1
		[E → • E – T]	1
		[T → • i]	3
		[T → • (E)]	4
1	[S → E • #]		5
	[E → E • + T]		6
	[E → E • - T		7
2	[E → T •]		R2
3	[T → i •]		R5
4	[T → (• E)]		10
		[E → • T]	2
		[E → • E + T]	10
		[E → • E – T]	10
		[T → • i]	3
		[T → • (E)]	4
5	[S → E # •]		Accept

We can construct an LR(0) parsing machine from the above item sets as

Click here for the complete LR(0) machine.



Using the above LR(0) machine, let's parse the sentence i – (i + i) #

Input Symbol	**N**	**S**	α	**A/π**
	-	0	i – (i + i) #	-

Grammar

1. S → E #
2. E → T
3. | E + T
4. | E - T
5. T → i
6. | (E)

Click here for the complete parsing sequence.

Not every grammar is LR(0). Consider the following example:

0. S → E #
1. E → E - T
2. | T
3. T → F ^ T
4. | F
5. F → (E)
6. | i

The LR(0) item sets for the above grammar are

State	Basis Set		Closure Set	Next State or Reduce
0	[S → • E #]			1
			[E → • E – T]	1
			[E → • T]	2
			[T → • F ^ T]	3
			[T → • F]	3
			[F → • i]	4
			[F → • (E)]	5
1	[S → E • #]			6
	[E → E • - T]			7
2	[E → T •]			R2
3	[T → F • ^ T]	**inadequate**		11
	[T → F •]	**state**		R4
4	[F → i •]			R6
5	[F → (• E)]			9
			[E → • E – T]	9
			[E → • T]	2
			[T → • F ^ T]	3
			[T → • F]	3
			[F → • (E)]	5
			[F → • i]	4
6	[S → E # •]			Accept
7	[E → E - • T]			8

		[T → • F ^ T]	3
		[T → • F]	3
		[F → • (E)]	5
		[F → • i]	4
8	[E → E – T •]		R1
9	[F → (E •)]		10
	[E → E • – T]		7
10	[F → (E) •]		R5
11	[T → F ^ • T]		12
		[T → • F ^ T]	3
		[T → • F]	3
		[F → • i]	4
		[F → • (E)]	5
12	[T → F ^ T •]		R3

We have a shift/reduce conflict in state 3. Hence, we need a more powerful LR parsing technique than LR(0) to parse sentences in the language of this grammar.

ANSWERS

N	**S**	α	**A**	π
3	0	a a c b b #	-	-
3	0 3	a c b b #	S	-
4	0 3 3	c b b #	S	-
5	0 3 3	b b #	R	3
6	0 3 ~~3~~ ~~5~~	b b #	R	2
5	0 3	b #	S	-
6	0 ~~3 5~~	b #	R	2
1	0 1	#	S	-
2	0 1	-	A	

Return

The complete item sets for the grammar are

State	Basis Set	Closure Set	Next State or Reduce
0	[S → • A #]		1
		[A → • a A b]	3
		[A → • c]	4
1	[S → A • #]		2
2	[S → A # •]		Accept
3	[A → a • A b]		5
		[A → • a A b]	3
		[A → • c]	4
4	[A → c •]		R3
5	[A → a A • b]		6
6	[A → a A b •]		R2

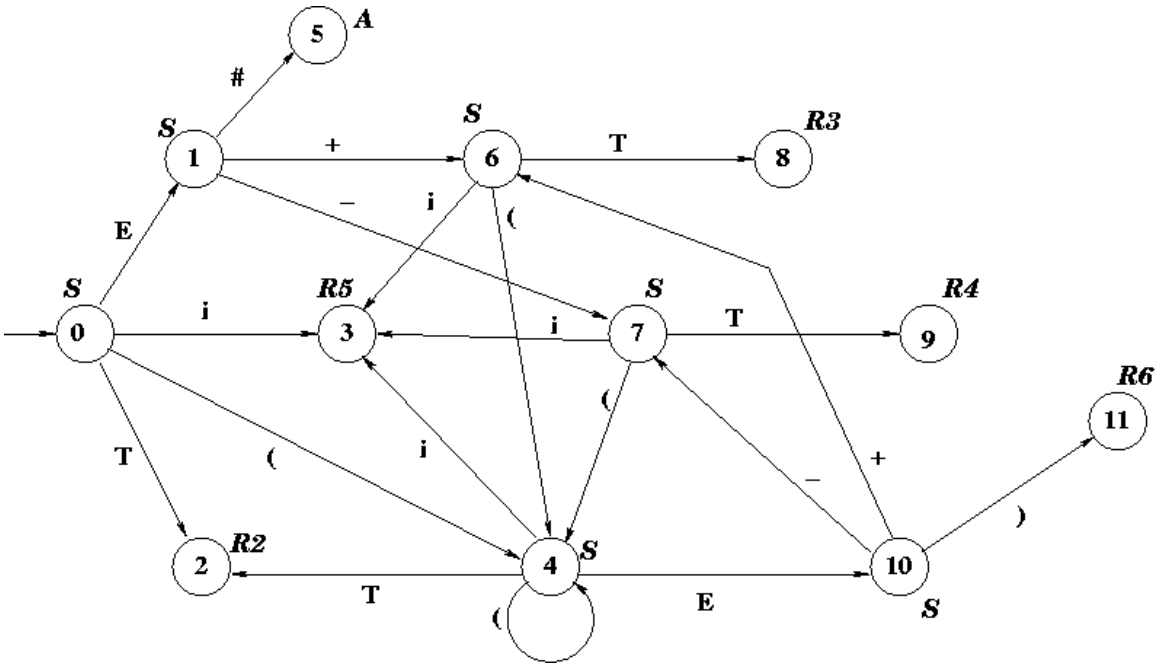
Return

The complete item sets for the grammar are

State	Basis Set	Closure Set	Next State or Reduce
0	[S → • E #]		1
		[E → • T]	2
		[E → • E + T]	1
		[E → • E – T]	1
		[T → • i]	3
		[T → • (E)]	4
1	[S → E • #]		5
	[E → E • + T]		6
	[E → E • - T		7
2	[E → T •]		R2
3	[T → i •]		R5
4	[T → (• E)]		10
		[E → • T]	2
		[E → • E + T]	10
		[E → • E – T]	10
		[T → • i]	3
		[T → • (E)]	4
5	[S → E # •]		Accept
6	[E → E + • T]		8
		[T → • i]	3
		[T → • (E)]	4
7	[E → E - • T]		9
		[T → • i]	3
		[T → • (E)]	4
8	[E → E + T •]		R3
9	[E → E – T •]		R4
10	[T → (E •)]		11
	[E → E • + T]		6
	[E → E • - T]		7
11	[T → (E) •]		R6

Return

The LR(0) machine for the above items sets is



In Figure above, we have reduce states 2, 3, 5, 8, 9, and 11 with the corresponding reductions to be made in those states. We have stack states 0, 1, 4, 6, 7, and 10.

Return

N	S	α	A	π
-	0	i – (i + i) #	-	-
3	0	– (i + i) #	R	5
2	0	– (i + i) #	R	2
1	0 1	– (i + i) #	S	-
7	0 1 7	(i + i) #	S	-
4	0 1 7 4	i + i) #	S	-
3	0 1 7 4	+ i) #	R	5
2	0 1 7 4	+ i) #	R	2
10	0 1 7 4 10	+ i) #	S	-
6	0 1 7 4 10 6	i) #	S	-
3	0 1 7 4 10 6	) #	R	5
8	0 1 7 4 ~~10 6~~	) #	R	3
10	0 1 7 4 10	) #	S	-
11	0 1 7 ~~4 10~~	#	R	6
9	0 ~~1 7~~	#	R	4
1	0 1	#	S	-
5	0 1	-	A	-

Return