LR Analysis: Exercise

Problem

- Repeat the analysis performed by the previous example, using the analysis table and input string shown in the next page.

60

LR Analysis: Exercise

State LRAnalyser

	Σ_T						Σ_N		
E	I	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			
	Action						Go-to		

(	+	id	)	$		


```
(1) E → E+T
(2)   |   T
(3) T → T*F
(4)   |   F
(5) F →(E)
(6)   |   id
```

)

61

LR Analysis: Exercise

	Σ_T						Σ_N		
E	I	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			
	Action						Go-to		

(	+	id	)	$		

0						

(1) E → E+T
(2) | T
(3) T → T*F
(4) | F
(5) F →(E)
(6) | id

62

LR Analysis: Exercise

	Σ_T						Σ_N		
E	I	+	*	(	)	$	E	T	F
0	s5			**s4**			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			
	Action						Go-to		

(	+	id	)	$		

4	id	0				

(1) E → E+T
(2) | T
(3) T → T*F
(4) | F
(5) F →(E)
(6) | id

63

LR Analysis: Exercise

	Σ_T						Σ_N		
E	I	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			
	Action						Go-to		

(	+	id	)	$		

4	id	0				

(1) E → E+T
(2) | T
(3) T → T*F
(4) | F
(5) F →(E)
(6) | id

64

LR(0) Analysis

Configuration, Analysis Element

- Intuitively, bottom-up analysis follows, simultaneously, all the possible analyses.
- Each possibility consists of marking, in the right-hand sides of the rules, the prefix that matches with the portion of the input string read.
- In other words, there will be different states for every possible shift situation.
- The **configuration**, also called **analysis element**, is the concept that formalises each one of the possibilities for analysis in the following way:
  - Let us imagine that the following is a rule of the grammar:

    E→ E+T
  - This rule can only be useful in the following cases:
    - When we are going to identify **E+T**
    - Having found (reduced) **E**, when we are going to identify **+T**
    - Having found **E+** (reduced **E** and shifted **+**), when we are going to identify **T**
    - Having found **E+T** (reduced **E**, shifted **+** and reduced **T**)

65

LR(0) Analysis

Representation of a configuration

- The representation of each of those positions can be stated more clearly, for instance, using the symbol •. If we number the productions, they can be represented with a pair as this:

  (<production no.>, <position in the right-hand side>)

- In this way, we would represent the previous example as follows: let us assume that the rule in the example is the j-th rule:
  - When we are going to identify E+T

    E→•E+T or **(j,0)**
  - Having found (reduced) E, when we are going to identify +T

    E→E•+T or **(j,1)**
  - Having found E+ (reduced E and shifted +), when we are going to identify T

    E→E+•T or **(j,2)**
  - Having found E+T (reduced E, shifted + and reduced T)

    E→E+T• or **(j,3)**

66

LR(0) Analysis

Configuration: types

- As shown previously, only when the complete right-hand side has been identified, a rule can be reduced.
- Therefore, we can consider two types of configurations:
  - **Reduction configurations**:
    - All these will have the dot at the end of the right-hand side.
    - With the other representation, the second numeric value of the pair will be equal to the length of the right-hand side of the rule.

      E→E+T• or **(j,3)**
  - **Shift configurations**:
    - All the remaining ones:

      E→E•+T or **(j,1)**

      E→E+•T or **(j,2)**

      E→E+T• or **(j,3)**

67

LR(0) Analysis

States of the automaton in an LR(0) analyser

- The configurations, or analysis elements, are grouped to form the states in the automata.
- The configurations are really the states of a non-deterministic finite automata (NFA), which recognises the so-called “viable prefixes”.
- The states of the equivalent DFA are sets of states of the NFA (**TALF I**).
- In order to build the states in the LR(0) analyser, the same process will be followed.

68

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

- The initial configuration of the analyser can be obtained from the axiom of the grammar.
- Let us consider the initial grammar in the example (the rules have been numbered for clarity):

```
(0) E'→E$
(1) E→      E+T
(2)   | T
(3) T→      i
(4)   |     (E)
```

- The initial configuration for analysis is:

  `E'→•E$` or `(0,0)`

  - This presents the following hypothesis: *if we manage to identify* `E$`, *we'll be able to reduce the input string to the axiom, and recognise it.* The position of the dot • indicates that we have not yet identified any of the symbols.

69

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

- Any time that the dot • precedes a non-terminal, we might we might find the coincidence with the input string in the derivations of the non-terminal. Therefore, without advancing in the input string, we should study the configurations that still have not started the right-hand sides of the non-terminal symbols. In this case:

  E→•E+T or **(1,0)**

  E→•T or **(2,0)**

- Remember that, in an NFA, λ-transitions can change the state of the automaton without consuming input symbols.
- Therefore, the first coincidence with the input may happen as a derivation of **E** or as a derivation of **T**.

  T→•i or **(3,0)**

  T→•(E) or **(4,0)**

70

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

- Putting all these configurations together, we have made the initial state, which we might call s_0:

  E'→•E$ or **(0,0)**

  E→•E+T or **(1,0)**

  E→•T or **(2,0)**

  T→•i or **(3,0)**

  T→•(E) or **(4,0)**

- From here on, we shall have to consider all possible situations to which we may go from this initial set of configurations, with any possible symbol from the alphabet of terminal ans non-terminal symbols.
- In other words, we shall study all the situations to which we may arrive, from the initial state, using the same method.

71

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

- For instance,
  - From s_0 with the symbol **E** we can go to a new state. In this case, with the **E**, just the two first configurations in s_0 (**E′→•E$** y **E→•E+T**) can change. In both cases, the dot would advance one symbol ahead:

    E′→E•$ or **(0,1)**

    E→E•+T or **(1,1)**
  - In both configurations, the dot • is before a terminal symbol. In these cases, we cannot obtain new configurations as we did before, because we cannot find new rules that could be applied at the points where the dot is located now.
  - In this case, we can go to a new state called s_1

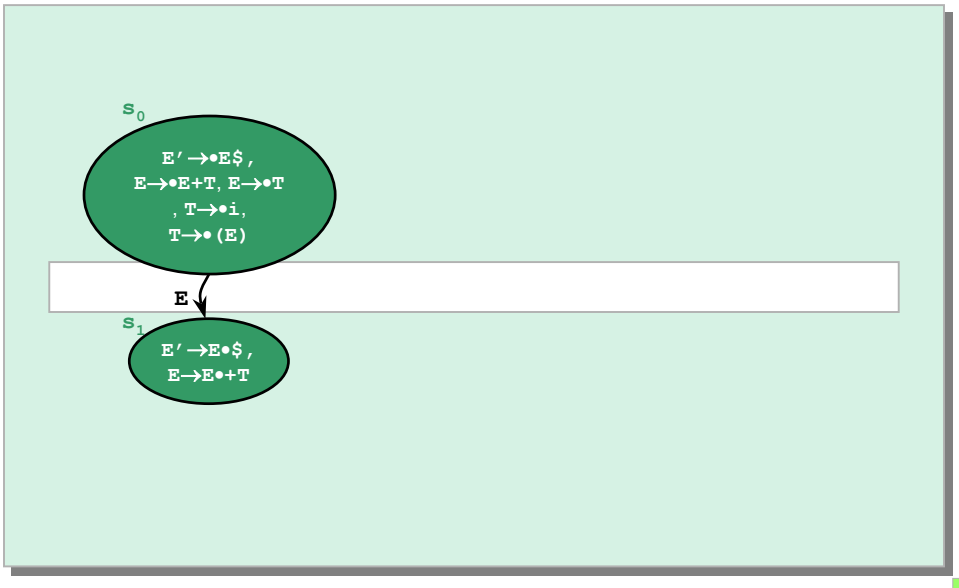
    E′→E•$ or **(0,1)**

    E→E•+T or **(1,1)**
- This transition can be represented graphically in a transition diagram:

72

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example



73

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

- And the process follows until there are no more sets of configuration which we have not analysed yet.
- Next, from s_1,

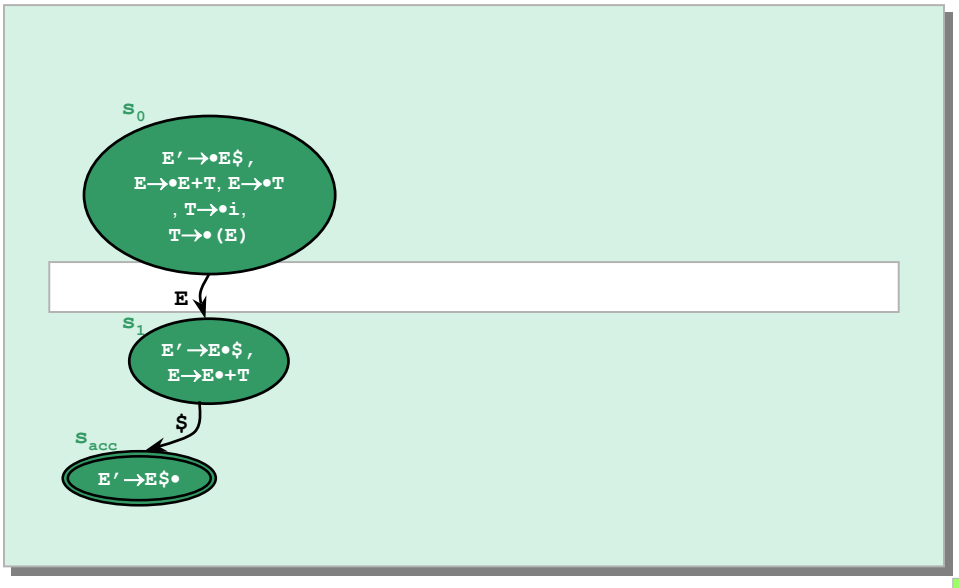
  s_1 ={E′→E•$ or (0,1), E→E•+T or (1,1)}
- We shall need to study if we can proceed from any of the configurations in s_1 with all the symbols in the alphabets.
- Let us take the first configuration E′→E•$. Each symbol which is different to the one with the dot "•" to its left will generate an empty set of configurations.
  - From E′→E•$ and $ we move to:
    - E′→E$• , which represents the hypothesis "having identified the complete input string". It is not possible to advance further.
    - Therefore, we'll move to a new state. Note that this is the only one which will accept the complete input string. We'll call it s_{acc}.
    - Note that it is a reduction configuration. In the diagram, all the reduction configurations will be represented as a final state.

s_{acc}={E′→E$• or (0,2)}

74

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example



75

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

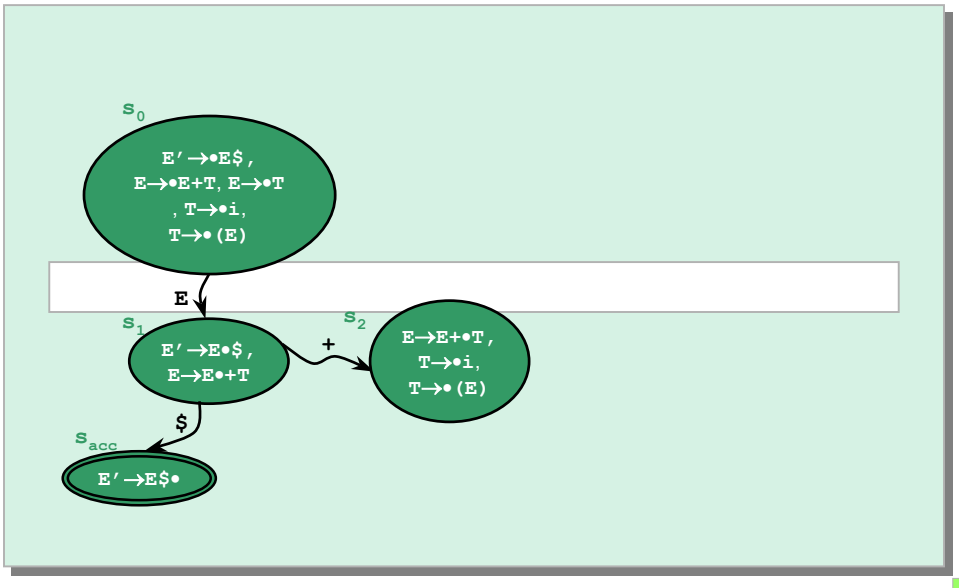
- From **E→E•+T** with + we would go to
  - **E→E+•T** , which represents the hypothesis of having identified (reduced) one **E**, shifted one **+**, and now expecting to identify (reduce) a non-terminal **T.**
  - If we expect to reduce a non-terminal, that is equivalent to expecting to identify any of the right-hand sides of the rules for that non-terminal. Therefore, with the "+" symbol, we can also go to the following configurations:
    - **T→•i** or **(3,0)**
    - **T→•(E)** or **(4,0)**
  - Those are all the configurations accessible from the state **E→E•+T** or **(1,1)**, if we find the symbol +. In summary, we would go to the following state:

S_2={**E→E+•T** or **(0,2)**, **T→•i** or **(3,0)**, **T→•(E)** or **(4,0)**}

76

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example



77

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

- We can continue from s_2
  - From **E→E+•T** or **(0,2)**, with the symbol **T**, to **E→E+T•** or **(0,3)**, which is a final state (s_3)
  - From **T→•i** or **(3,0)** with **i** we go to **T→i•** or **(3,1)**, final state (s_4)
  - From **T→•(E)** or **(4,0)** and **(** we go to:
    - **T→(•E)** or **(4,1)**. Because **E** is not a terminal, this implies:
    - **E→•E+T** or **(1,0)**
    - **E→•T** or **(2,0)**. Because **T** is not a terminal, this implies:
    - **T→•i** or **(3,0)**
    - **T→•(E)** or **(4,0)**
  - If we call this new state s_5, it will be defined as:

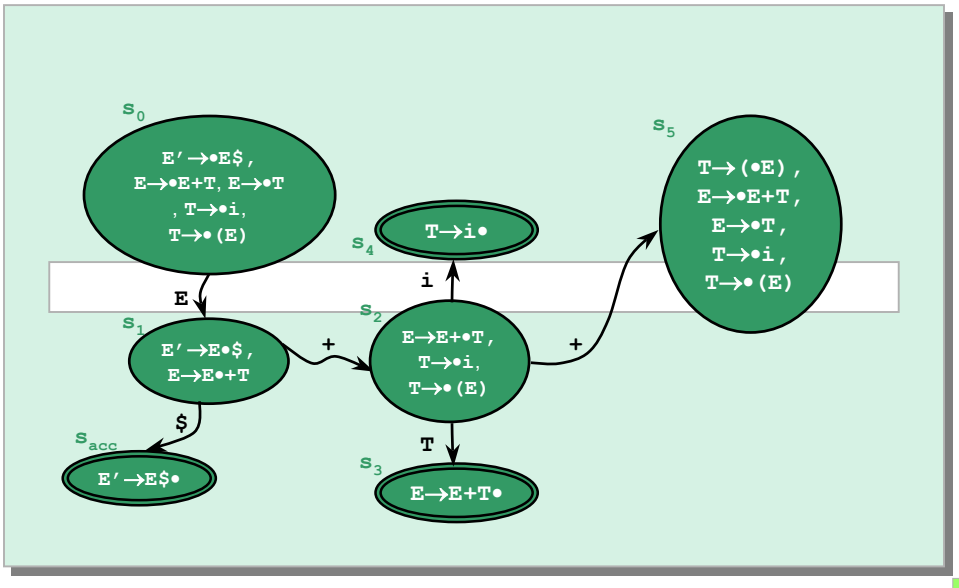
$$s_5 = \{T \rightarrow (\bullet E)\ o\ (4,1),\ E \rightarrow \bullet E+T\ o\ (1,0),\ E \rightarrow \bullet T\ o\ (2,0),\ T \rightarrow \bullet i\ o\ (3,0),\ T \rightarrow \bullet (E)\ o\ (4,0)\}$$

  - From s_2 it is not possible to reach any other set of configurations.

78

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example



79

LR(0) Analysis

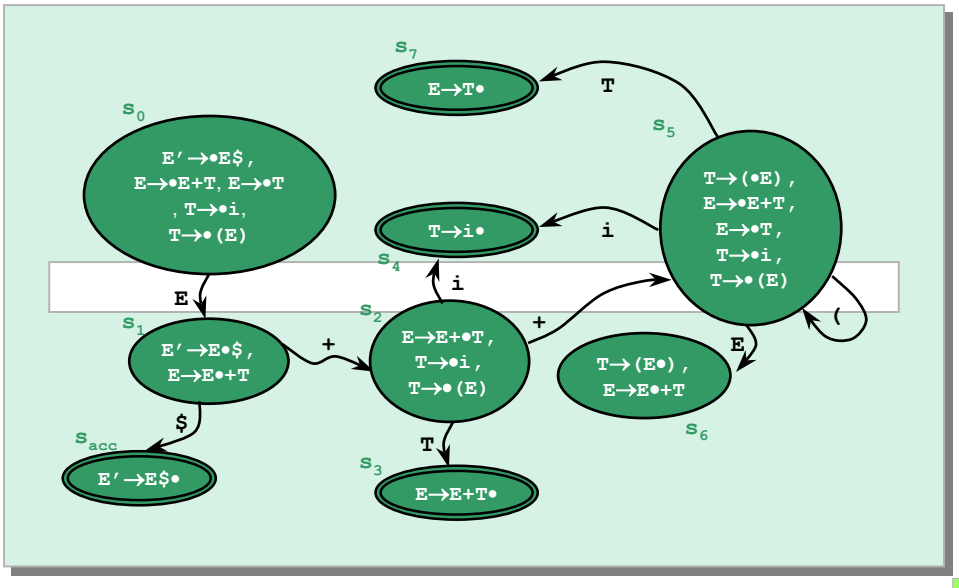
States of the automaton in an LR(0) analyser: introductory example

- From the final states we cannot reach any other node. Therefore, we may focus now on s_5
  - With an **E**,
    - From **T→(•E)** or **(4,1)** we can go to **T→(E•)** or **(4,2)**
    - From **E→•E+T** or **(1,0)** we can go to **E→E•+T** or **(1,2)**

      s_6={**T→(E•)** or **(4,2)**, **E→E•+T** or **(1,2)**}
  - From **E→•T** or **(2,0)** and **T**, we can go to **E→T•** or **(2,1)** (s_7 ,final)
  - From **T→•i** or **(3,0)** and **i**, we can go to **T→i•** or **(3,1)** (s_4)
  - From **T→•(E)** or **(4,0)** and **(**, we can go to **T→(•E)** or **(4,1)**. This state was called before s_5
  - From this state, we cannot reach any other

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example



LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example

- Let us now analyse the possibilities from the state (s_6).

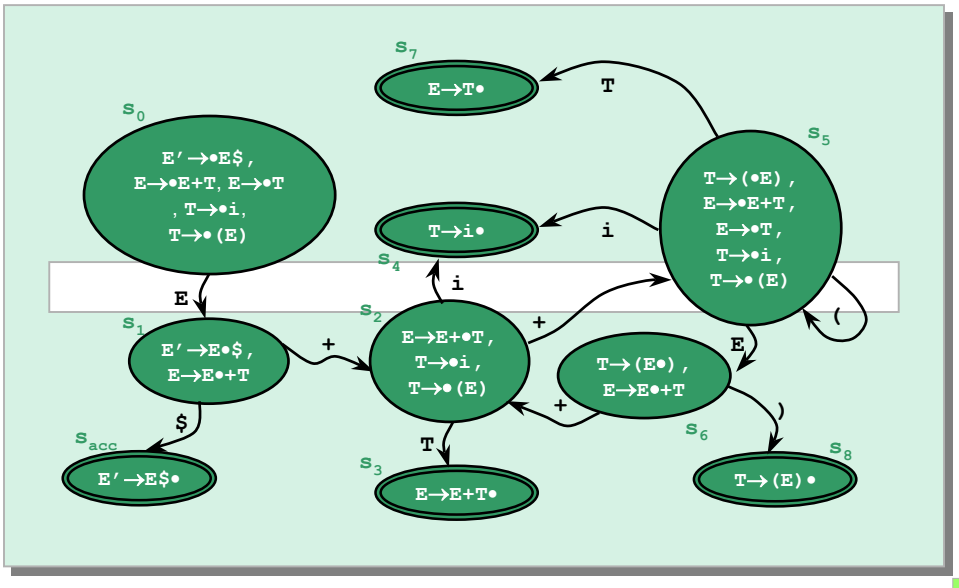
  s_6={T→(E•) or (4,2), E→E•+T or (1,2)}

  - From T→(E•) or (4,2) with), we can only go to T→(E)• or (4,3) which is a new state we shall call s_8 (final)
  - From E→E•+T or (1,2) and +, we can go to E→E+•T and the configurations that can be obtained from the T, state that we already know as s_2
  - We cannot reach any other state from here.

82

LR(0) Analysis

States of the automaton in an LR(0) analyser: introductory example



83

LR(0) Analysis

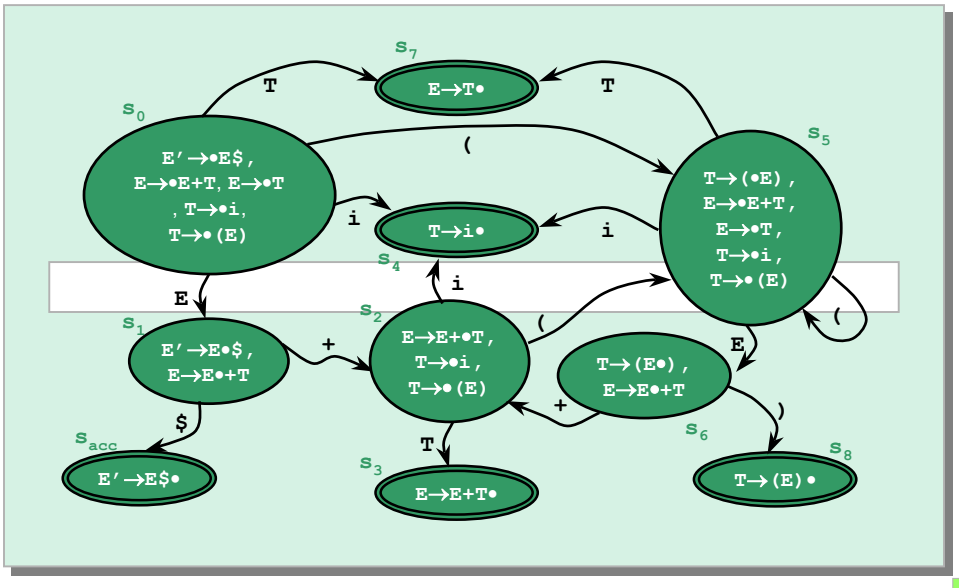
States of the automaton in an LR(0) analyser: introductory example

- By analysing this diagram, we can see that we still have to follow some possibilities from (s_0), as we only considered the analysis of the non-terminal symbol **E**.
- We can study all the transitions from s_0 with all the remaining symbols:
  - From **E→•T** or **(2,0)** and **T**, we can go to **E→T•** or **(2,1)** which is s_7
  - From **T→•i** or **(3,0)** and **i**, we can go to **T→i•** or **(3,1)** which is s_4
  - From **T→•(E)** or **(4,0)** and **(**, we can go to **T→(•E)** or **(4,1)** and the configurations that we can obtain from it, which is s_5
  - There are no more possibilities available.

84

LR(0) Analysis

States of the automaton in an LR(0) analyser: FINAL DIAGRAM



85

LR(0) Analysis

States of the automaton in an LR(0) analyser: formalisation

- The transition diagram obtained is the one that will be followed in the transition table that we have seen in the previous examples.
- In order to study the algorithm for LR(0) parsing more formally, it will be necessary to define a few additional concepts:
  - The **Augmented grammars**
  - The *Closure* operation.
  - The *Go-to* operation.
  - The **Canonical collection** of LR(0) configurations.

86

LR(0) Analysis

States of the automaton in an LR(0) analyser: **augmented grammar**

- **Formally,**
  - Given a grammar:

$$G = \langle \Sigma_N, \Sigma_T, P, E \rangle$$

  - The augmented grammar is:

$$G' = \langle \Sigma_N, \Sigma_T, P \cup \{E' \rightarrow E\$\}, E \rangle$$

- **Intuitively**
  - We have just seen that the addition of the new rule does not change the language.
  - Some of the algorithms we shall see require that that rule exist.

87

LR(0) Analysis

States of the automaton in an LR(0) analyser: **closure operation**

- **Formally,**
  - Let `I` be a set of configurations, referred to the augmented grammar `G'` defined before:

    We define the set `closure(I)` as the set containing:

    1. $\forall\ c \in I \Rightarrow c \in closure(I)$
    2. $(A \to \alpha \bullet B\beta) \in closure(I) \wedge (B \to \gamma) \in P \Rightarrow (B \to \bullet\gamma) \in closure(I)$

- **Intuitively,**
  - This operation allows us to group the analysis elements in sets that behave equally with respect to being accessible from other states. In this way, each of these sets is equivalent to an state in the associated automaton.

88

LR(0) Analysis

States of the automaton in an LR(0) analyser: **closure operation**

- The following algorithm calculates the closure of a configuration

```
ConfigurationSet Closure(ConfigurationSet I,
                         ContextIndependentGrammar Gic)

{
  ConfigurationSet Closure := I;
  Configuration c;
  ProductionRule r;

  while( "Closure has been extended with new configurations" )
  {
        "Repeat, for each c in Closure, for each r in Rules(Gic) "
        /* Assume that c is A→α•Bβ and r B→γ */
        if (B→•γ ∉ Closure)
                Closure := Closure ∪ {B→•γ };
  }
  return Closure;
}
```

89

LR(0) Analysis

States of the automaton in an LR(0) analyser: **go-to operation**

- **Formally**

  - Let `I` be a set of configurations, and `X` a symbol in the grammar `G'` (terminal or non-terminal),

  We define the operation `go-to(I,X)` as

  $$\text{union}\{\text{closure}(A \to \alpha X \bullet \beta)\},\ \forall\ A \to \alpha \bullet X\beta \in I$$

- **Intuitively,**
  - This is the operation that allows us to calculate, for each state in the automaton of the LR analyser, which is the state to which we can go.
  - In this way, we can finish calculating the diagram.

90

LR(0) Analysis

States of the automaton in an LR(0) analyser: **canonical configurations**

- **Formally,**

  - If we have the grammar `G'`, as defined before, the set of **canonical configurations LR(0)** are defined as follows:

  $\text{closure}(E \to E' \$)$ is canonical LR(0) (the initial state)

  If `I` is canonical LR(0), then

  $\text{Go\_to}(I,X)\ \forall X \in \Sigma_N \cup \Sigma_T$ is also canonical LR(0) $\Leftrightarrow$ $\text{go\_to}(I,X) \neq \emptyset$

- **Intuitively,**
  - This is the method used in the example.

91

LR(0) Analysis

States of the automaton in an LR(0) analyser: **canonical configurations**

- The following pseudocode can be used:

```
SetOfStates CanonicalStatesLR(0)(ContextIndependentGrammar Gic)
{
  ConfigurationsSet s[];
  integer i,j,k;

  i:=0;
  s[i]:=closure({axiom(Gic)'→axiom(Gic) $}); /* S' -> S$ */

  "repeat for each j≤i"
  {
        "repeat for each element X in Σ_N∪Σ_T  "
        {
              if ( (go_to(s[j],X)≠∅) ∧
                   (∀ k ∈ [0,i] s[k]≠go_to(s[j],X) )
                 )
                    i++;
                    s[i]=go_to(s[j],X);
        }
        j++;
  }
  return s;
}
```

92

LR(0) Analysis

Method for constructing LR(0) tables

- We shall distinguish the part of the table which indicates **shifts,** and the one that identifies **reductions**:
  - **Shifts in the table:**
    - Obtained by "reading the transitions in the automaton".
    - In other words, if the automaton goes from s_i to s_j with the symbol (terminal o not) **X**, then we add the following action to the cell:
      - `Syntactic_table[i,X]=`

$$\begin{cases} Sj & Si\ X \in \Sigma_T \\ j & Si\ X \in \Sigma_N \end{cases}$$

  - **Reductions in the table:**
    - For each terminal and each state with reduction configurations (of the type $A \rightarrow \gamma\bullet$) we have to add the reduction $A \rightarrow \gamma$.

93

LR(0) Analysis

Method for constructing LR(0) tables

- **Acceptation:**
  - If a state s_i has a transition, with the terminal `$`, to the state with the configuration `axioma'→axioma$•`, then we have to add the action **accept** to `Syntactic_table[i,$]`.
- **Error:**
  - All the others cells have the associated value `error`
  - The most usual procedure is to leave those cells blank, so we can interpret a blank cell as an `error` condition.

This is not the only procedure for building LR(0) parsers, so it is possible to find similar approaches.

94