Syntax Analysis
Earley Algorithm

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

A GENERAL METHOD FOR SYNTAX ANALYSIS: EARLEY

The EARLEY method (or algorithm) allows to deal with any grammar, even with an ambiguous one, as it constructs all the possible derivations of the string to recognise.

Its time complexity is proportional to the third power of the string length, but is lower if the grammar is unambiguous, and even less if it is deterministic.

The Earley method originates from the LR(*k*) method, but it does not attempt to build a deterministic pushdown automaton; rather it exploits a data structure more complex than the stack is (actually it uses an array of sets), which represents efficiently many possible stacks, sharing some contents.

In the practice, the Earley algorithm simulates a non-deterministic pushdown automaton, but without falling into the exponential time complexity that indeterminism usually implies.

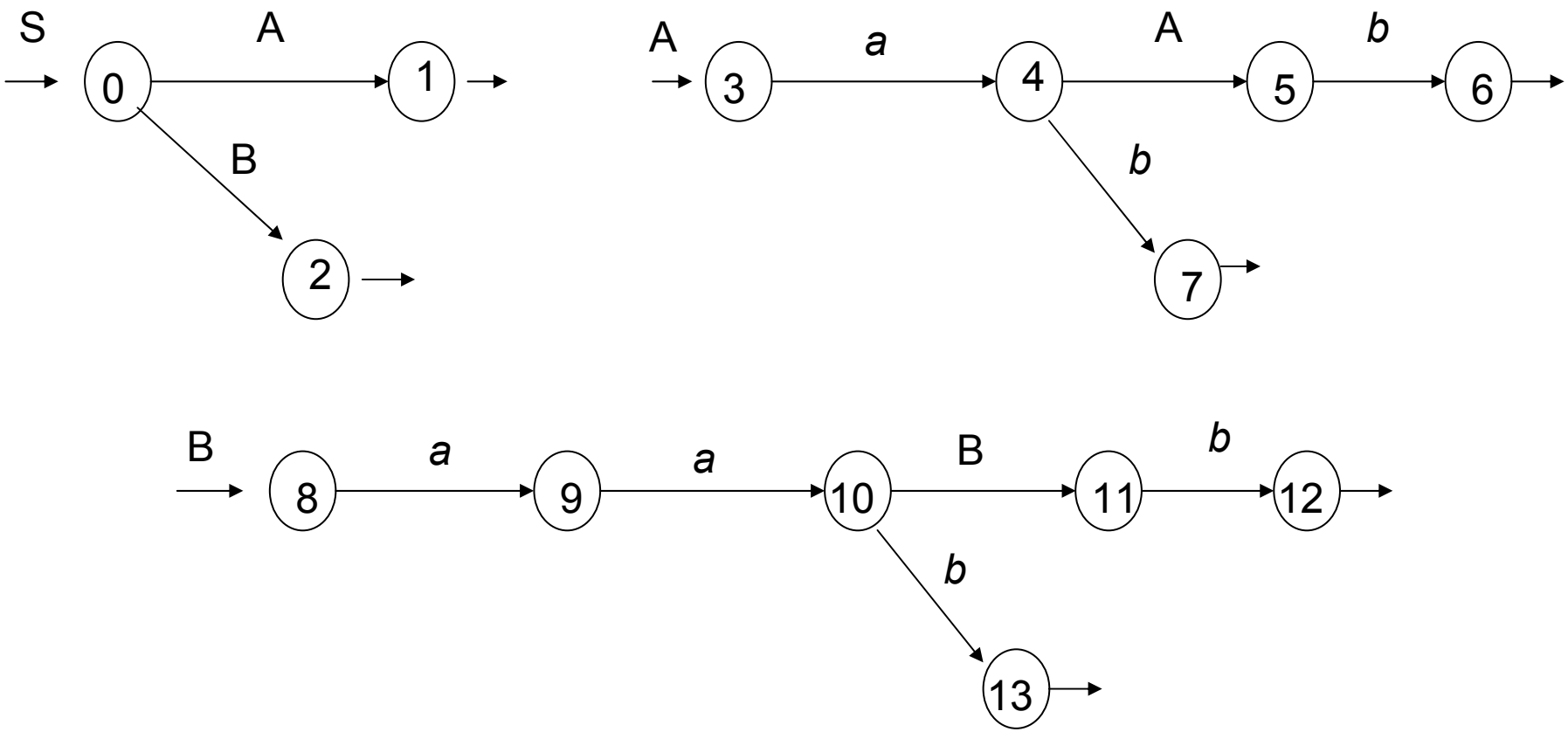
STARTING HYPOTHESIS: THE GRAMMAR DOES NOT CONTAIN NULL RULES

The Earley algorithm can work with or without lookahead, but the asymptotic time complexity is not influenced. For simplicity here lookahead is not considered.

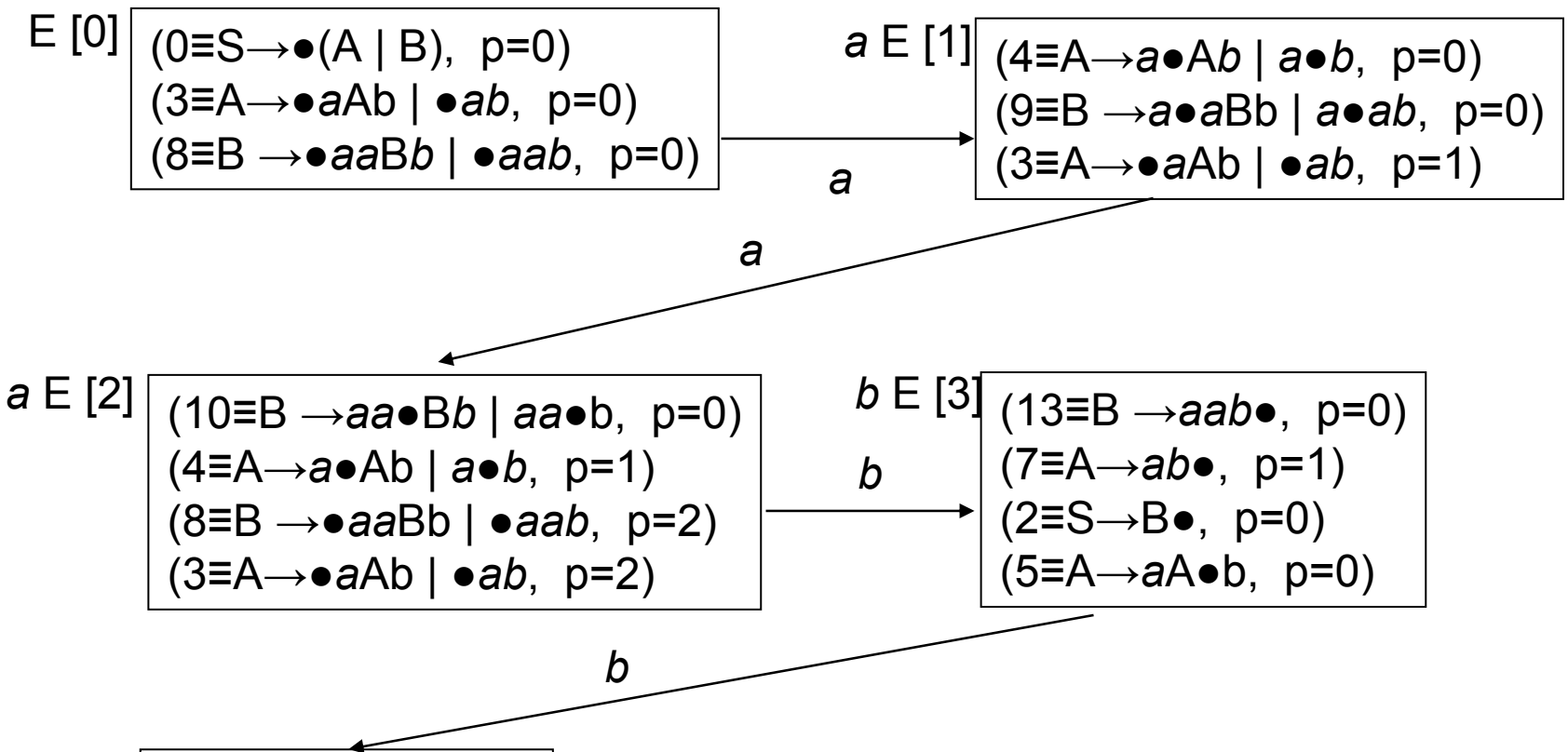
EXAMPLE - a non-LR(*k*) grammar

$$L = \left\{ a^n b^n \mid n \geq 1 \right\} \bigcup \left\{ a^{2n} b^n \mid n \geq 1 \right\}$$

$$S \to A \mid B \quad A \to aAb \mid ab \quad B \to aaBb \mid aab$$



string to analyse: *aabb*



Supposing the string to analyse is *aab,* it would be accepted as in macrostate E [3] there is a pair (state = 2, position = 0). But the complete string is *aabb*, and is accepted in E [4] as such a macrostate contains the pair (state = 1, position = 0).

The Earley algorithm is a top-down analyzer that constructs simultaneously all the possible left derivations of the string. The algorithm reads the string $x_{1\ldots n}$ from left to right and, when it examines character x_i, produces pairs of two elements, as follows:

<*machine state, position in the string*> denoted as (s = ..., p = …) with $0 \le p \le i$

As already seen, the state can be alternatively denoted as marked rule s ≡ A→α ● β

Intuitively pair (s ≡ A→α ● β, p) represents a <u>statement</u> and a <u>target</u>:

<u>Statement</u>: there has been found a substring $x_{j+1\ldots i}$ with $0 \le j < i$ that derives from α

<u>Target</u>: find all the positions $k > i$ such that substring $x_{i+1\ldots k}$ derives from β

If the algorithm finds such a position k, it can state that from non-terminal A there derives substring $x_{i+1\ldots k}$

$$\alpha \overset{*}{\Rightarrow} x_{j+1\ldots i}$$

$$\beta \overset{*}{\Rightarrow} x_{i+1\ldots k}$$

$$A \overset{*}{\Rightarrow} x_{j+1\ldots k}$$

A pair (q ≡ A→ α ●, j), where state q is final (end-marked rule), is said to be <u>completed.</u>

ALGORITHM – Earley recognizer

The task of the Earley recognizer is that of finding the derivation of the entire string *x*, but the algorithm generally produces more than it is strictly requested: it also finds whether any prefix *u* of the string $x = uv$ belongs to the language itself.

The Earley algorithm builds a vector E [0…*n*] (n is the string length) dimensioned to the string length, the elements of which are pairs.

PASS 1: *Initialization* - set the targets to find all the prefixes of *x* that are derivable from the axiom S. The initial set is filled with pairs obtained from the axiom S and all the other sets are initially empty.

$$E[0] := \{(q_\alpha, 0) \mid q_\alpha \text{ is the network initial state}\}$$

$$E[i] := 0, \quad \text{for } i = 1, \ldots, n$$

$$i := 0$$

PASS 2: *Derivation Construction* - then the operations of prediction, completion and shift are applied in the natural order 0, 1, …, *n*, in order to compute all the sets E [*i*].

At step *i* the algorithm can add elements only to the current set E [*i*] and to the consecutive one E [*i* + 1].

If none of the operations has added new pairs to E [*i*], the algorithm moves to consider set E [*i* + 1].

The algorithm stops and rejects the string if E [*i* + 1] is empty and $i < n$.

The algorithm terminates with success and accepts the string when the set E [*n*] has been fully filled and contains (at least) one completed pair ($q \equiv S \rightarrow \alpha \bullet, 0$) of the axiom (clearly *q* is the network initial state).

PREDICTION OPERATION (CLOSURE)

Every target in the set E [*i*] can add new sub-targets to the set itself. Whenever a new sub-target is added, its position pointer is set to the current position *i*.

for every pair ($q \equiv A \rightarrow \alpha \bullet B\ \gamma$, j) already in E [*i*]
add to the set E [*i*] the new pair ($r \equiv B \rightarrow \bullet \ldots$, i)
where *r* is the initial state of the machine expanding B

SHIFT OPERATION

Update the targets in the set E [*i* + 1] depending on the next character in the input string. The position pointer of the updated targets is the same as that of the current target.

for every pair ($q \equiv A \rightarrow \alpha \bullet a\ \gamma$, j) already in E [*i*]
if $x_{i+1} = a$ then
add to the set E[*i* + 1] the new pair ($r \equiv A \rightarrow \alpha\ a \bullet \gamma$, j)

Each character x_{i+1} of the input string is examined by the algorithm only once.

COMPLETION OPERATION

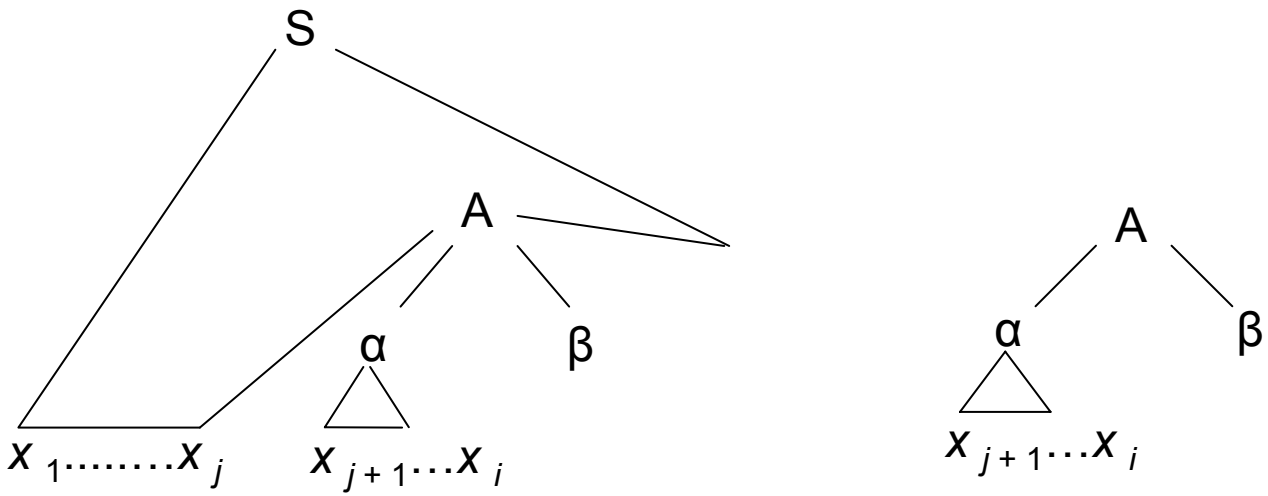
A completed pair ($q \equiv A \rightarrow \alpha \bullet, j$) states that the derivation of string $x_{j+1 \ldots i}$ has been identified starting from the non-terminal A. It is necessary to update set E [i] and add to it such a statement.

for every completed pair ($q \equiv A \rightarrow \alpha \bullet, j$) already in E [$i$]
for every pair ($r \equiv B \rightarrow \beta \bullet A\ \gamma$, k) already in E [$j$]
add to the set E [i] the new pair ($s \equiv B \rightarrow \beta\ A \bullet \gamma, k$)

On completion, if pair ($r \equiv B \rightarrow \beta \bullet A\ \gamma, k$) $\in$ E [j] and the string γ is null, the new pair ($s \equiv B \rightarrow \beta\ A \bullet \gamma, k$), just added to the set E [i], is itself completed, and therefore it is necessary to iterate the completion operation again.

It is easy to prove that the Earley algorithm accepts only strings belonging to the language L(G): a new pair is added to a set only if the derivation that it states is possible in G. It is more complex to prove that every phrase of the language is recognised by the algorithm (proof omitted here).

HOW TO RECONSTRUCT THE SYNTAX TREE PROGRESSIVELY



It is instructive to imagine the three operations as a progressive construction of the syntax tree: to each array set E there corresponds a set of trees.

In E [*i*] there is a pair (s ≡ A→α ● β, p = *j*) if and only if grammar G admits a syntax tree as shown above (left tree). The right tree shows the specific subtree associated with the pair.

At the end of the algorithm, the string is accepted if and only if one of the trees associated with the last set E [*n*], is a complete syntax tree with axiomatic root.

E [0]

(0≡S→●(A | B), p=0)
(3≡A→●*a*Ab | ●*ab*, p=0)
(8≡B →●*aa*Bb | ●*aab*, p=0)

a E [1]

(4≡A→*a*●A*b* | *a*●*b* ,p=0)
(9≡B →*a*●*a*B*b* | *a*●*ab*, p=0)
(3≡A→●*a*A*b* | ●*ab*, p=1)

a

a E [2]

(10≡B →*aa*●B*b* | *aa*●*b*, p=0)
(4≡A→*a*●A*b* | *a*●*b*, p=1)
(8≡B →●*aa*B*b* | ●*aab*, p=2)
(3≡A→●*a*A*b* | ●*ab*, p=2)

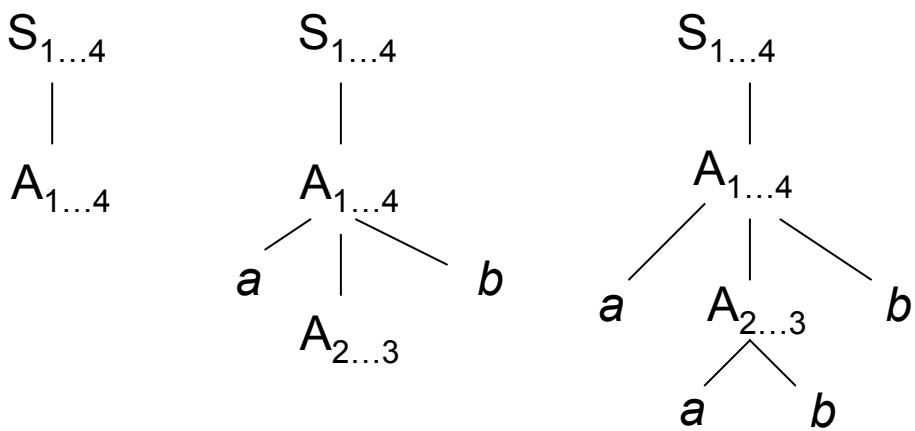
b E [3]

(13≡B →*aab*●, p=0)
(7≡A→*ab*●, p=1)
(2≡S→B●, p=0)
(5≡A→*a*A●*b*, p=0)

b

b E [4]

(6≡A→*a*A*b*●, p=0)
(1≡S→A●, p=0)



If the grammar is ambiguous, analysing a phrase produces all the possible syntax trees, represented by sharing common factors.

EXAMPLE – parsing an ambiguous language – the grammar contains a two-sided recursive rule, hence is ambiguous

phrase *a* + *a* + *a*

$$S \to E \qquad E \to E + E \qquad E \to a$$

E [0]

S→●E, 0
E→●E+E | ●*a*, 0

a E [1]

E→a●, 0

S→E●, 0
E→E●+E, 0

+ E [2]

E→E+●E, 0
E→●E+E | ●*a*, 2

a E [3]

E→*a*●, 2

E→E+E●, 0
E→E●+E, 2

S→E●, 0
E→E●+E, 0

+ E [4]

E→E+●E, 0
E→E+●E, 2
E→●E+E | ●*a*, 4

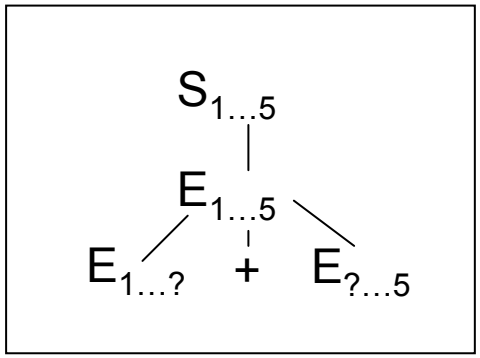
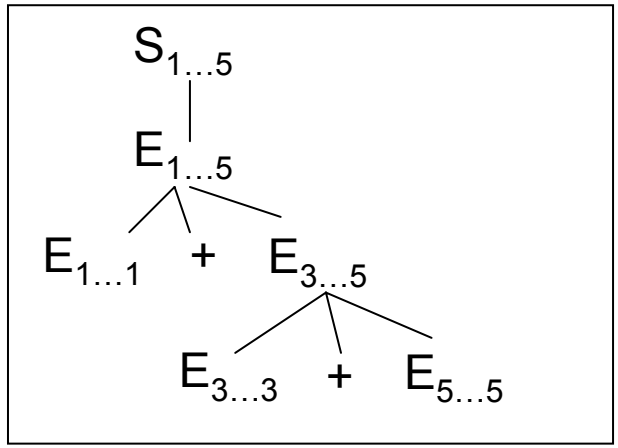
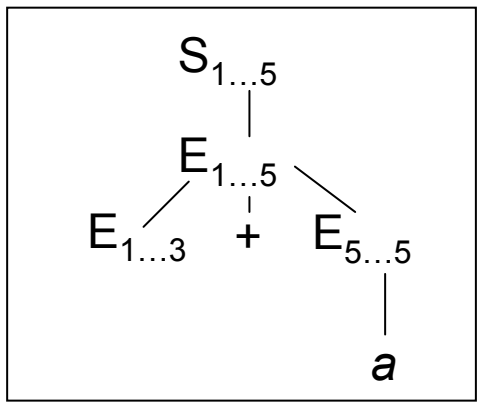
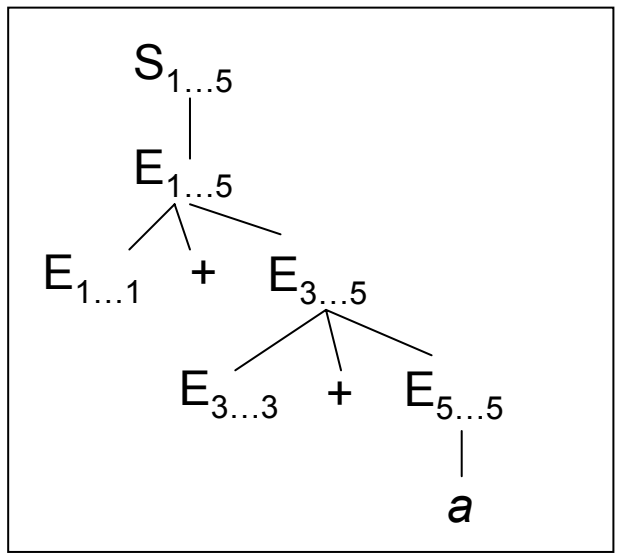
a E [5]

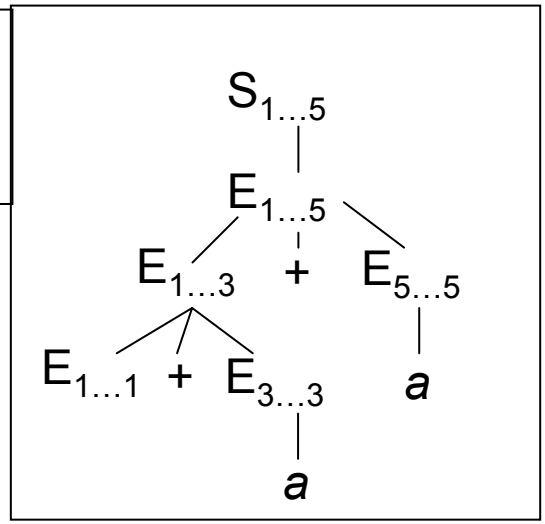
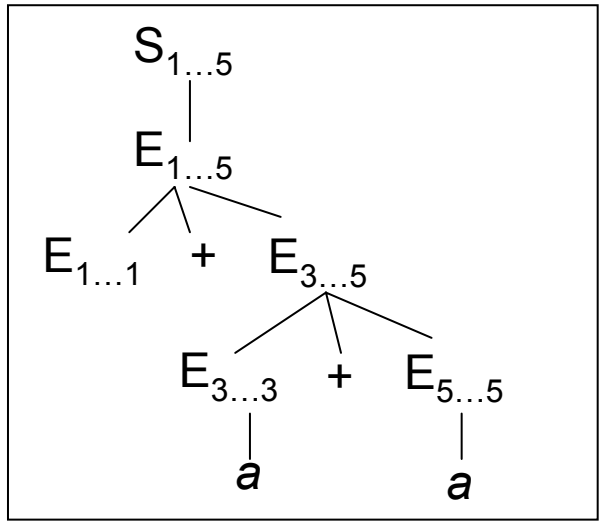
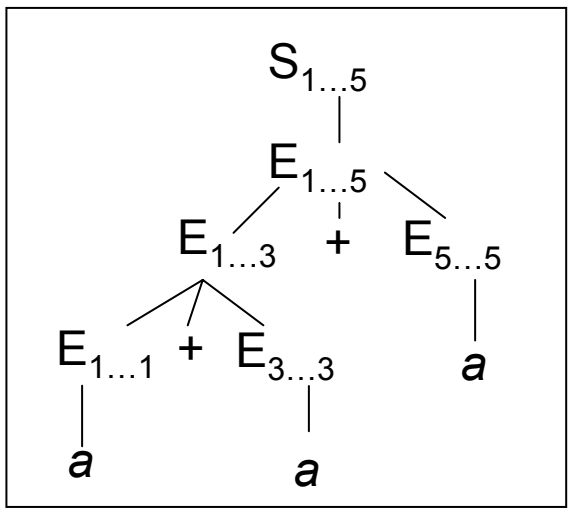
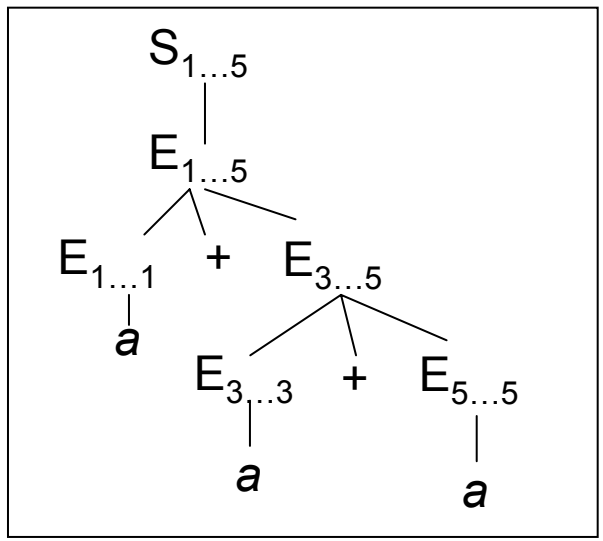
E→*a*●, 4

E→E+E●, 2
E→E●+E, 4

E→E+E●, 0
E→E●+E, 2

S→E●, 0
E→E●+E, 0

Pair	1st Tree	Pair	2nd Tree
E [5] S→E●, 0 E→E+E●, 0		E [5] S→E●, 0 E→E+E●, 2	
E→*a*●, 4		E→*a*●, 4	

Pair	1st Tree	Pair	2nd Tree
E [3] E→E+E●, 0 E→*a*●, 2		E [3] E→*a*●, 2	
E [1] E→*a*●, 0		E [1] E→*a*●, 0	

HOW TO DEAL WITH NULL RULES

The Early algorithm, as it has been illustrated so far, builds two sets of pairs: E [*i*], which holds the pairs produced by predition and completion, and E [*i* + 1], which holds pairs produced by shift.

In the presence of ε-rules, the completion operation should examine set E [*i*] built so far, and then invoke prediction, which should in turn invoke completion, and so on until the two operations do not have anything left to add to the set. This method is correct, but inefficient.

The *Aycock* and *Horspool* method is more efficient and modifies the prediction operation as follows:

PREDICTION OPERATION (with ε-rules)

for every pair ($q \equiv A \rightarrow \alpha \bullet B\ \gamma$, *j)* already in E [*i*]
add to the set E [*i*] the new pair ($r \equiv B \rightarrow \bullet\ \delta$, *i*)
where *r* is the initial state of the machine expanding B
if B is a nullable non-terminal
add to the set E [*i*] also the new pair ($s \equiv A \rightarrow \alpha\ B \bullet \gamma$, *j*)

The new fact is that the marker ● is moved to the right side of the nullable non-terminal symbol B, in agreement with the observation that the derivation might end up with replacing the non-terminal B by ε.

EXAMPLE – a grammar where all non-terminal symbols are nullable

$$S' \to S \quad S \to AAAA \quad A \to a \quad A \to E \quad E \to \varepsilon$$

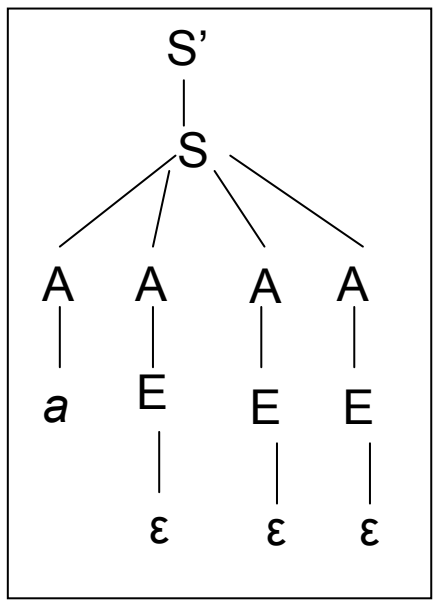
phrase to recognise: *a*

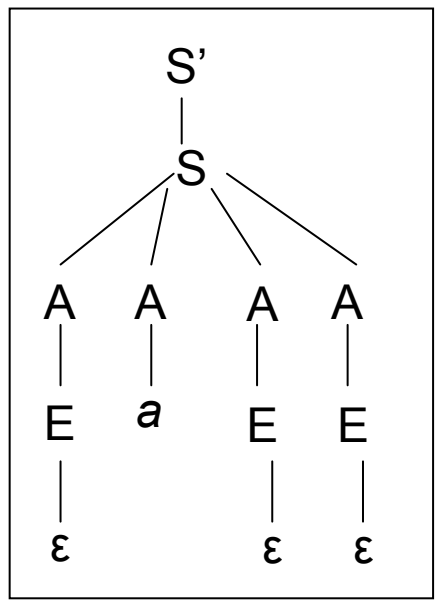
E [0]

S'→●S, 0
S→●AAAA, 0
S'→S●, 0
A→●*a*, 0
A→●E, 0
S→A●AAA, 0
E→●, 0
A→E●, 0
S→AA●AA, 0
S→AAA●A, 0
S→AAAA●, 0

a E [1]

A→*a*●, 0
S→A●AAA, 0
S→AA●AA, 0
S→AAA●A, 0
S→AAAA●, 0
A→●a, 1
A→●E, 1
S' →S●, 0
E→●, 1
A→E●, 1





and so on …..

IMPROVEMENTS OF THE EARLEY METHOD

1) HOW TO COMBINE EARLEY AND LOOKAHEAD

The pairs contained in the Earley sets can be extended by adding lookahead information as well. Lookahead sets are computed as in the LR(1) case.

If pairs are associated with lookahead sets, the Earley algorithm can avoid to add to a set a pair that is certainly doomed to failure. However, adding lookahead may sometimes cause the number of pairs to increase (as each pair contains more information). The effective advantage of adding lookahead to Earley is still a controversial matter.

The most efficient implementations of the Earley method do not use lookahead.

2) HOW TO WORK EARLEY OUT WITH EBNF GRAMMARS

The Earley algorithm can be modified so as to work correctly with EBNF rules (actually, representing a grammar by means of a machine network does so).

Bibliography

- S. Crespi Reghizzi, *Linguaggi Formali e Compilazione*, Pitagora Editrice Bologna, 2006
- Hopcroft, Ullman, *Formal Languages and their Relation to Automata*, Addison Wesley, 1969
- A. Salomaa – *Formal Languages*, Academic Press, 1973
- D. Mandrioli, C. Ghezzi – *Theoretical Foundations of Computer Science*, John Wiley & Sons, 1987
- L. Breveglieri, S. Crespi Reghizzi, *Linguaggi Formali e Compilatori: Temi d'Esame Risolti,* web site (eng + ita)