Syntax Analysis of Non-Deterministic Grammars Earley (Tabular) Method

Translated and adapted by L. Breveglieri

A GENERAL SYNTAX ANALYSIS METHOD – EARLEY (TABULAR)

The *Earley* method (also called *tabular* method) deals with any grammar type, even ambiguous, but we fully apply it only to non-deterministic grammars

It builds in parallel all the possible derivations of the string prefix scanned so far

Earley is similar to *ELR*, but it does not use the stack; instead, it uses a vector of sets, which efficiently represents stacks that have common parts

It resembles closely the implementation of the *ELR* deterministic *PDA* by means of a vectored stack

It simulates a non-deterministic pushdown analyzer, but with a polynomial time complexity

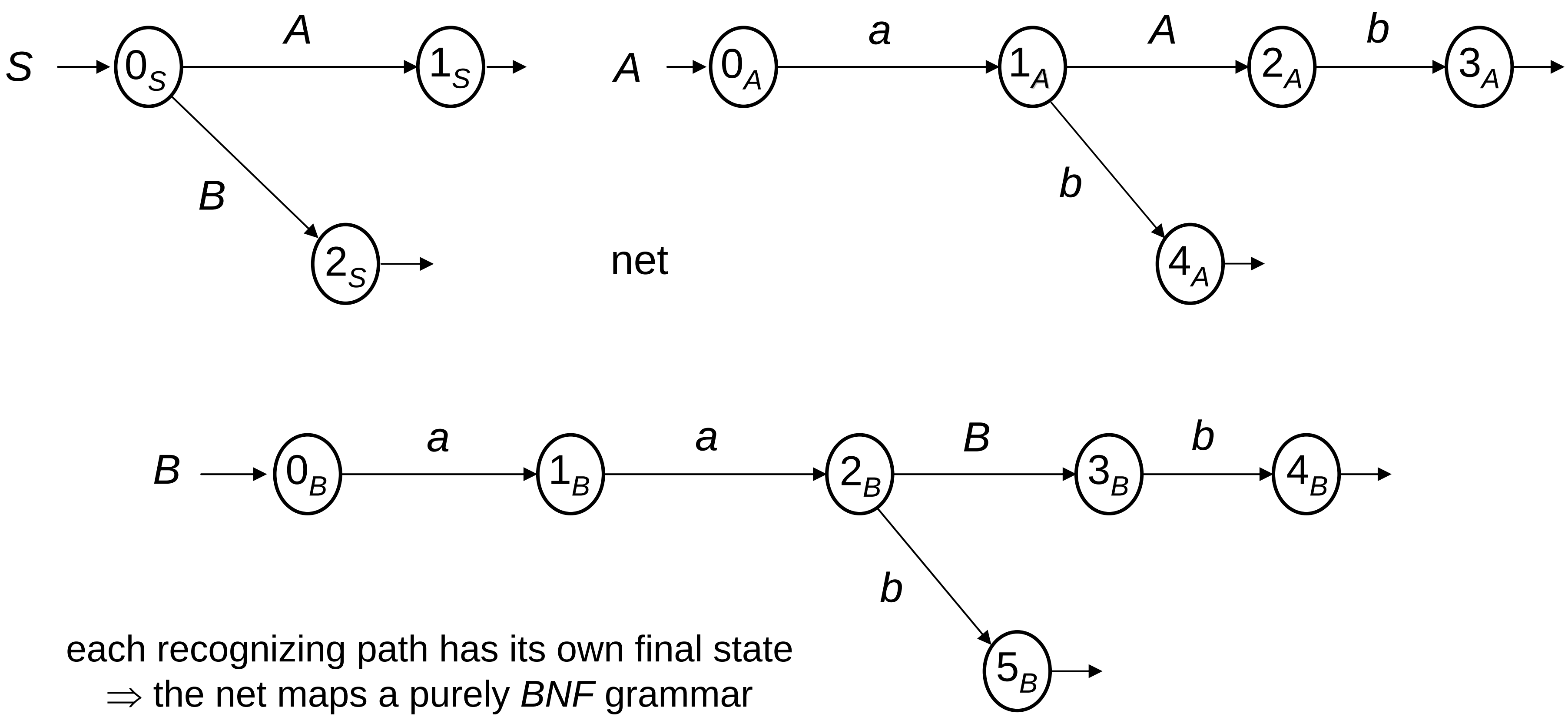
Notice that also the *ELR* method simulates parallel analysis threads, but only until reduction time

The Earley algorithm has variants without or with look-ahead, but here for simplicity look-ahead is not used

EXAMPLE – a grammar not *LR* (*k*)

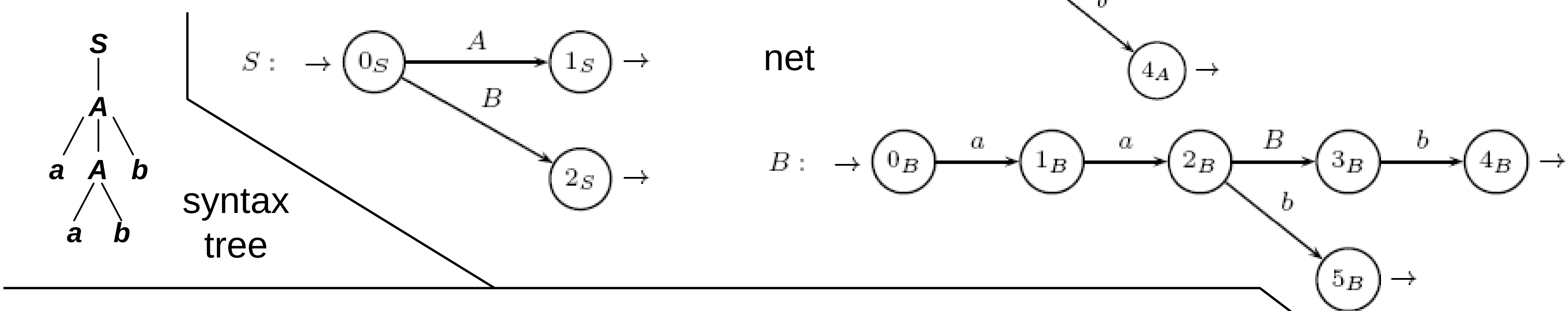
$$L = \left\{ a^n b^n \mid n \geq 1 \right\} \cup \left\{ a^{2n} b^n \mid n \geq 1 \right\}$$

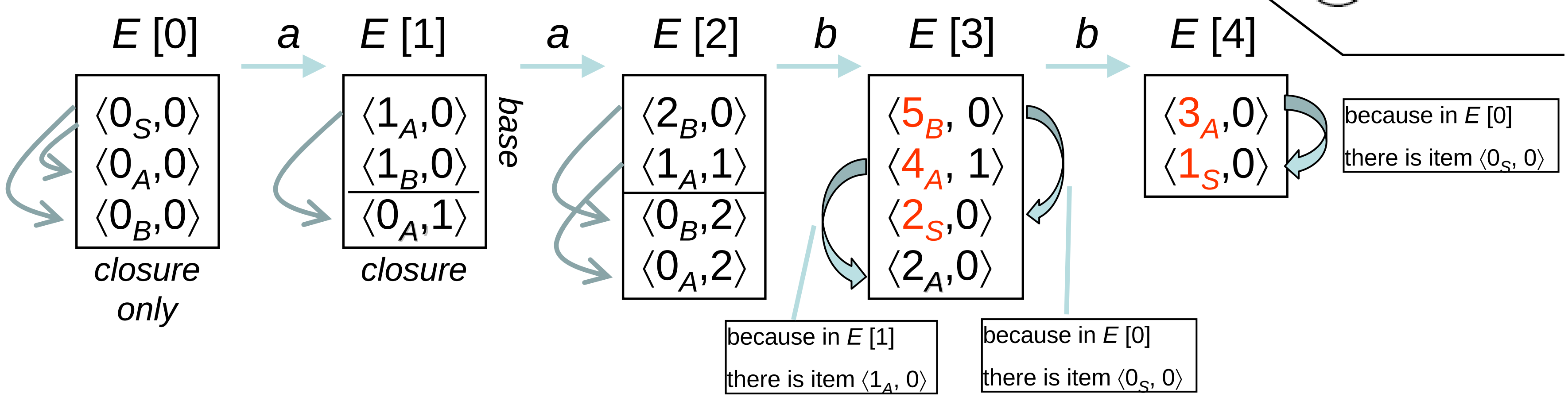
$$S \to A \mid B \qquad A \to aAb \mid ab \qquad B \to aaBb \mid aab$$



each recognizing path has its own final state
$\Rightarrow$ the net maps a purely *BNF* grammar

ANALYSIS – string *“a a b b”*





a = terminal shift   = closure   = non-terminal shift

final states are in red

If the string were *“a a b”*, it would be accepted because $\langle 2_S, 0\rangle \in E\,[3]$ and 2_S is final for M_S

But it is the string “*a a b b*” that is accepted, because $\langle 1_S, 0\rangle \in E\,[4]$ and 1_S is final for M_S

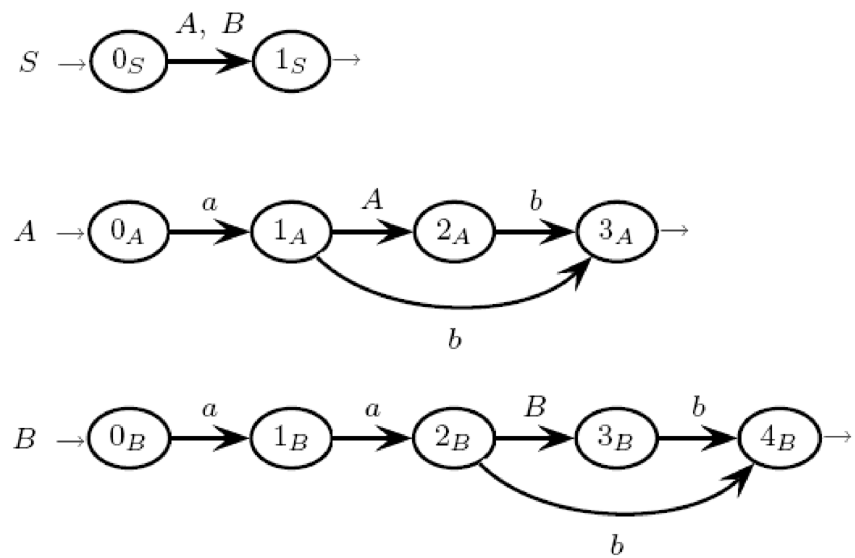
Besides deciding if it holds $x \in L$, the algorithm decides also for all the prefixes of string x

SAME LANGUAGE AS BEFORE BUT WITH A NOT *ELR* (*k*) GRAMMAR

$S \rightarrow A \mid B$

$A \rightarrow a\ A\ b \mid a\ b$

$B \rightarrow a\ a\ B\ b \mid a\ a\ b$



grammar still *BNF*
but the final states
are unified

Earley vector $E[0 \ldots 4]$

$E(0)$		x_1	$E(1)$		x_2	$E(2)$		x_3	$E(3)$		x_4	$E(4)$	
q	j	a	q	j	a	q	j	b	q	j	b	q	j
0_S	0		1_A	0		2_B	0		4_B	0		3_A	0
0_A	0		1_B	0		1_A	1		3_A	1		1_S	0
0_B	0		0_A	1		0_B	2		1_S	0			
						0_A	2		2_A	0			

ALGORITHM – the Earley (tabular) recognizer

The algorithm builds a vector $E\ [0 \ldots n]$ of $n = |\ x\ |$ elements

Each element $E\ [i]$ is a set of items (or pairs) $\langle\ s, j\ \rangle$ like the items in the *LR* stack, but without look-ahead

The vector E contains all the analysis threads in a compacted form, i.e., the common parts are shared

We define the move types that are found in the steps of the algorithm

terminal shift	scanning: there are at most n shifts, one per each character in x
closure	same as the closure in *LR*
non-terminal shift	same as the non-terminal shift in *LR*
completion	$=$ closure $+$ non-terminal shift, possibly repeated two or more times

call *i* the *current position*, the max index value *i* such that the set *E* [*i*] is not empty

TerminalShift $(E,\ i)$ -- with index $1 \leq i \leq n$

-- loop that computes the *terminal shift* operation
-- for each preceding pair that shifts on terminal x_i

for $\left(\text{each pair } \langle\, p,\ j \,\rangle \in E\,[i-1] \text{ and } q \in Q \text{ s.t. } p \xrightarrow{x_i} q\right)$ **do**

add pair $\langle\, q,\ j \,\rangle$ to element $E\,[i]$

end for

-- analyze the terminal string x for possible acceptance
-- define the Earley vector $E\,[0 \ldots n]$ with $|x| = n \geq 0$

$E\,[0] := \{\, \langle\, 0_S,\ 0 \,\rangle \,\}$ -- initialize the first elem. $E\,[0]$

for $i := 1$ **to** n **do** -- initialize all elem.s $E\,[1 \ldots n]$

$E\,[i] := \emptyset$

end for

Completion $(E,\ 0)$ -- complete the first elem. $E\,[0]$

$i := 1$

-- while the vector is not finished and the previous elem. is not empty

while $(\, i \leq n \ \wedge\ E\,[i-1] \neq \emptyset \,)$ **do**

TerminalShift $(E,\ i)$ -- put into the current elem. $E\,[i]$

Completion $(E,\ i)$ -- complete the current elem. $E\,[i]$

$i{+}{+}$

end while

Earley

Completion $(E,\ i)$ -- with index $0 \leq i \leq n$

do

-- loop that computes the *closure* operation
-- for each pair that launches machine M_X

for $\left(\text{each pair } \langle\, p,\ j \,\rangle \in E\,[i] \text{ and } X,\ q \in V,\ Q \text{ s.t. } p \xrightarrow{X} q\right)$ **do**

add pair $\langle\, 0_X,\ i \,\rangle$ to element $E\,[i]$

end for

closure

-- nested loops that compute the *nonterminal shift* operation
-- for each final pair that enables a shift on nonterminal X

for $(\text{each pair } \langle\, f,\ j \,\rangle \in E\,[i] \text{ and } X \in V \text{ such that } f \in F_X)$ **do**

-- for each pair that shifts on nonterminal X

for $\left(\text{each pair } \langle\, p,\ l \,\rangle \in E\,[j] \text{ and } q \in Q \text{ s.t. } p \xrightarrow{X} q\right)$ **do**

add pair $\langle\, q,\ l \,\rangle$ to element $E\,[i]$

end for

end for

non-terminal shift

while (some pair has been added)

every string character x_{i+1} is examined only once in the scanning
the machine input head is one-way and there is not backtracking.

string *x* is accepted if and only if item $\langle\, f_S,\ 0 \,\rangle \in E\,[n]$ $\quad f_S \in F_S$ is a final state for machine *S*

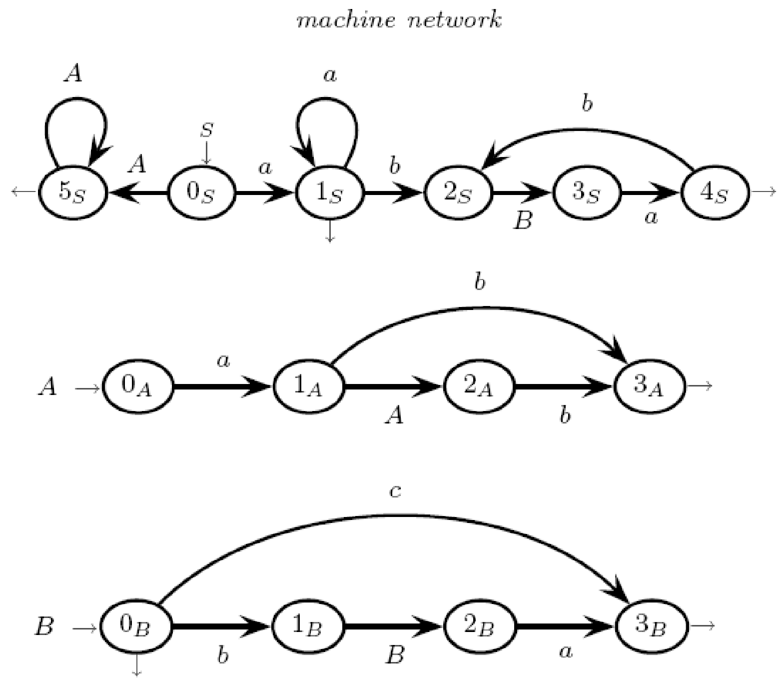
MORE COMPLEX EXAMPLE WITH A NON-DETRMINISTIC GRAMMAR (not *ELR* (1))

ext. grammar (EBNF)

$S \rightarrow a^+ \; (b \, B \, a)^* \; | \; A^+$

$A \rightarrow a \, A \, b \; | \; a \, b$

$B \rightarrow \; b \, B \, a \; | \; c \; | \; \varepsilon$

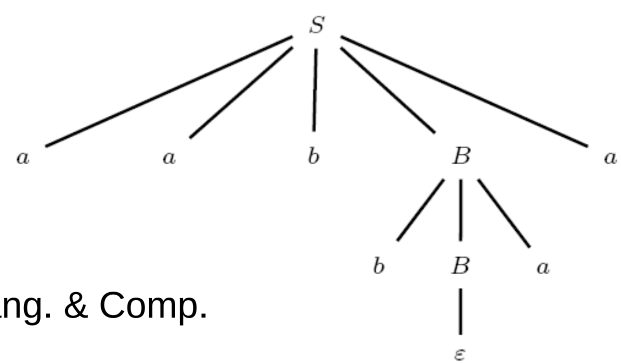


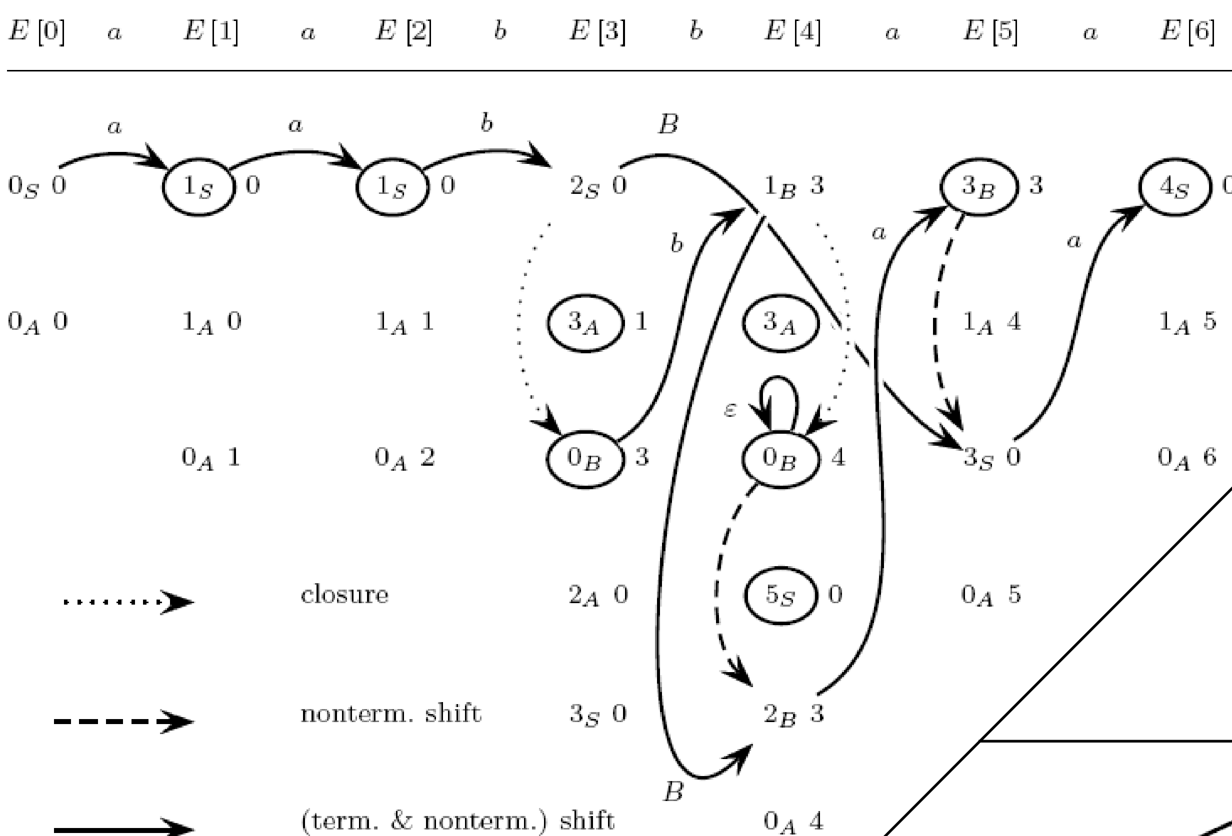
grammar is *EBNF*
$\Rightarrow$ its net has loops

NON-DETERMINISM: if
- the adjacent substrings of *a* and *b* have the same length
- there is not any character *c*

Then the string deep structure can be determined only at the end of the scanning

syntax tree of string
"*a a b b a a*"

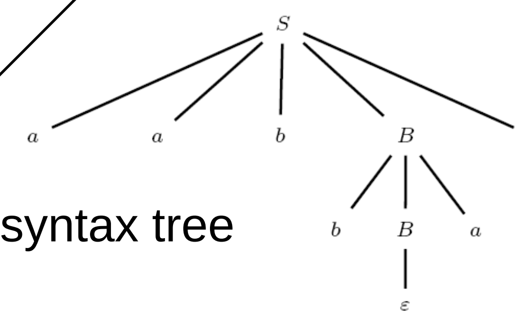




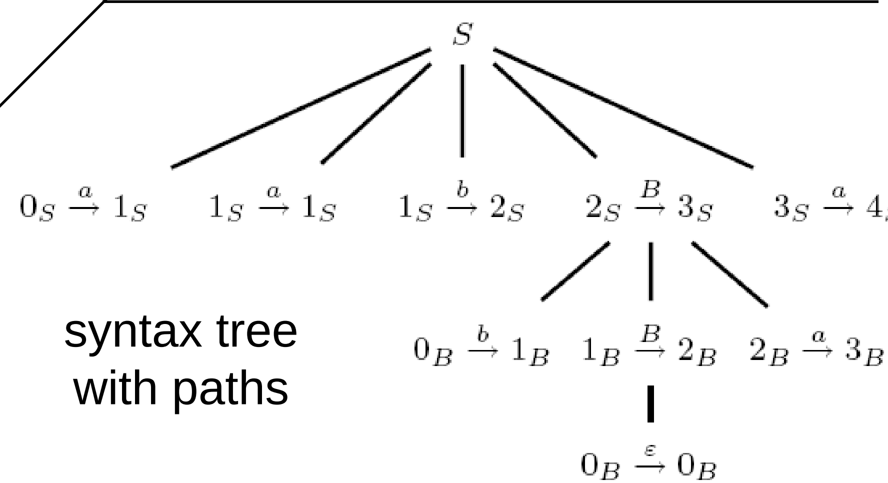
ANALYSIS

includes the case of a nullable non-terminal

syntax tree



syntax tree with paths



the grammar (or the net)
is not ambiguous
$\Rightarrow$ only one tree can be identified

WORST-CASE ASYMPTOTIC COMPLEXITY OF EARLEY

Let $n \geq 1$ be the length of the string to be analyzed

space complexity $=$ size of the Earley vector

$=$ # of sets E [j] $\times$ max size of an element
$= (n + 1) \times kn$ — k is the number of net states
$= O\,(n^2)$ — is quadratic

time compexity $=$ # of terminal shifts $+$ # of closures $+$ # of nonterminal shifts

$= O\,(n^2) + O\,(n^2) + O\,(n^2) \times kn$ — k is the number of net states
$= O\,(n^3)$ — is cubic

So in general Earley is polynomial, though it is not linear

Special case: $n = 0$ (null string), then both complexities are constant, as only closure is used and is limited by the number of states k, which is a constant; anyway, the complexity is still polynomial

In most practical cases with applicative grammars (or machine nets), Earley performs better than above and approaches a linear behaviour

RECONSTRUCTION OF THE SYNTAX TREE
(for non-ambiguous grammars only)

The recursive procedure *BuildTree* (*BT*) builds the one tree starting from vector E

BT has four parameters:

- nonterminal X, subtree root, with $X \in V$
- final state of machine M_X
- integer i, substring left position
- integer j, substring right position

BT returns: the (sub)tree of the derivation $X \Rightarrow^* x_{i+1}, \dots x_j$ that has root X

Initial call $BT(S, f_S, 0, n)$

INITIALIZATION AND MAIN OPERATIONS

$BuildTree\ (\ X,\ f,\ j,\ i\ )$

- - X is a nonterminal, f is a final state of M_X and $0 \leq j \leq i \leq n$
- - return as parenthesized string the syntax tree rooted at node X

- - node X will have a list $\mathcal{C}$ of terminal and nonterminal child nodes
- - either list $\mathcal{C}$ will remain empty or it will be filled from right to left

$\mathcal{C} := \varepsilon$ - - set to ε the list $\mathcal{C}$ of child nodes of X

$q := f$ - - set to f the state q in machine M_X

$k := i$ - - set to i the index k of vector E

In its recostruction loop, function *BT* performs the following operations:

- scanning the vector *E* backwards by iterating the while loop
- rebuilding terminal and non-terminal shift moves
- identifying the child nodes of the same parent node
- recursively calling itself if set *E*[*i*] contains a pair from non-terminal shift

$\Rightarrow$ the recursive call builds the subtree of the shifted non-terminal

SYNTAX TREE RECONSTRUCTION LOOP

- - walk back the sequence of term. & nonterm. shift oper.s in M_X

while ($q \neq 0_X$) **do** - - while current state q is not initial

- - try to backwards recover a terminal shift move $p \xrightarrow{x_k} q$, i.e.,
- - check if node X has terminal x_k as its current child leaf

(a) **if** $\left(\begin{array}{l} \exists\, h = k - 1 \;\; \exists\, p \in Q_X \;\; \text{such that} \\ \langle\, p,\ j\, \rangle \in E\,[h] \;\wedge\; \text{net has } p \xrightarrow{x_k} q \end{array} \right)$ **then**

$\mathcal{C} := x_k \cdot \mathcal{C}$ - - concatenate leaf x_k to list $\mathcal{C}$

end if

terminal shift

- - try to backwards recover a nonterm. shift oper. $p \xrightarrow{Y} q$, i.e.,
- - check if node X has nonterm. Y as its current child node

(b) **if** $\left(\begin{array}{l} \exists\, Y \in V \;\; \exists\, e \in F_Y \;\; \exists\, h \;\; j \leq h \leq k \leq i \;\; \exists\, p \in Q_X \;\; \text{s.t.} \\ \langle\, e,\ h\, \rangle \in E\,[k] \;\wedge\; \langle\, p,\ j\, \rangle \in E\,[h] \;\wedge\; \text{net has } p \xrightarrow{Y} q \end{array} \right)$ **then**

- - recursively build the subtree of the derivation:
- - $Y \overset{+}{\underset{G}{\Rightarrow}} x_{h+1} \ldots x_k$ if $h < k$ or $Y \overset{+}{\underset{G}{\Rightarrow}} \varepsilon$ if $h = k$
- - and concatenate to list $\mathcal{C}$ the subtree of node Y

$\mathcal{C} := \mathit{BuildTree}\ (\, Y,\ e,\ h,\ k\,) \cdot \mathcal{C}$

end if

non-terminal shift

$q := p$ - - shift the current state q back to p

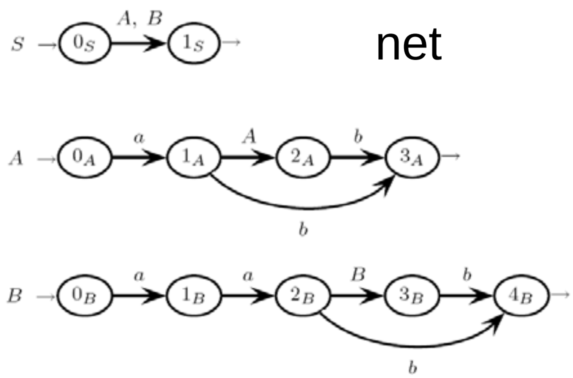
$k := h$ - - drag the current index k back to h

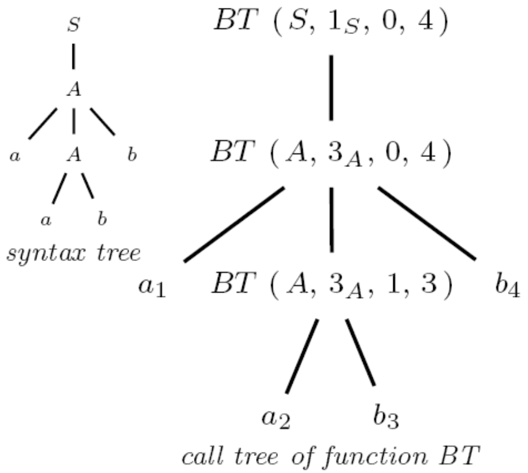
end while

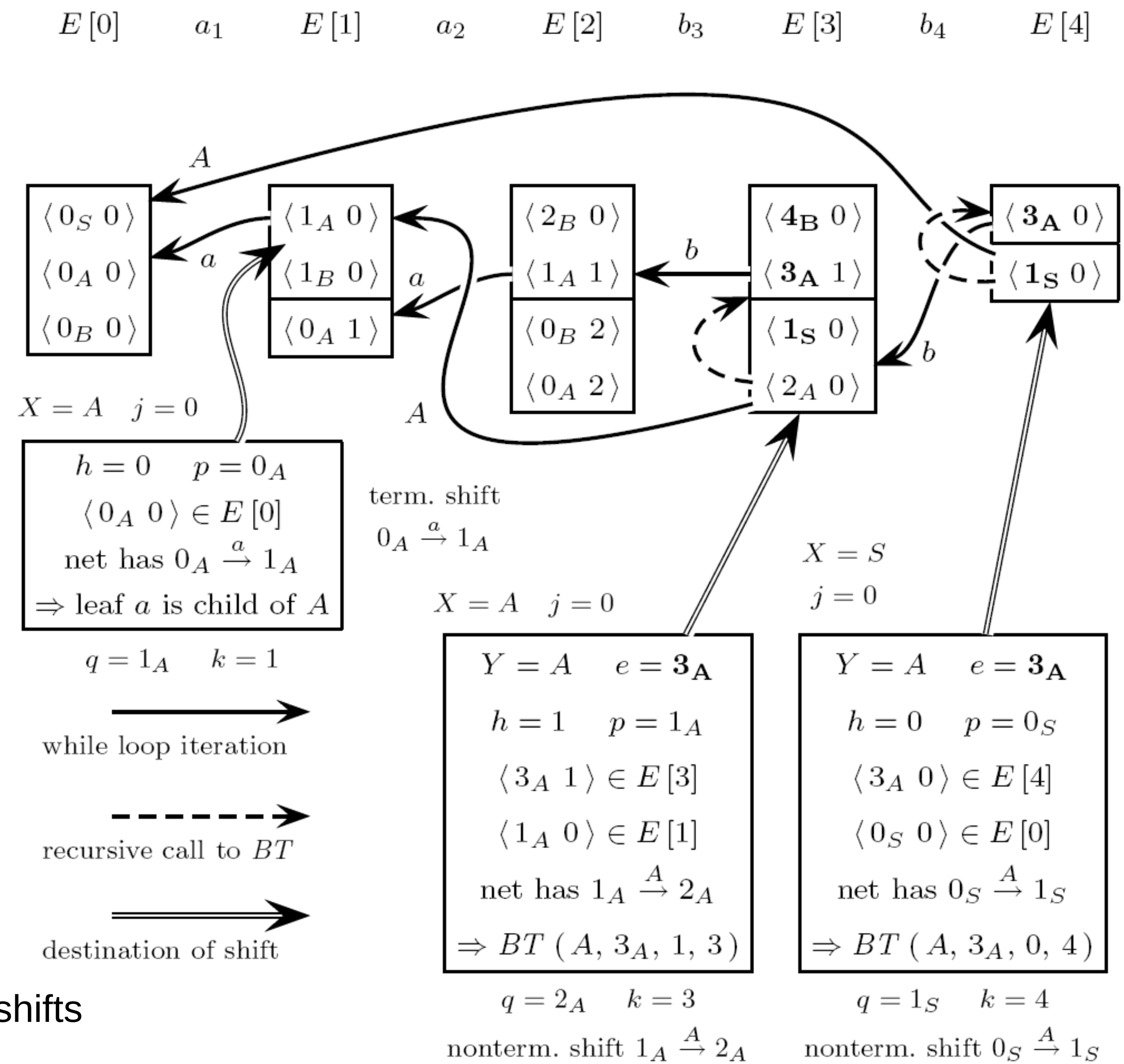
return $(\, \mathcal{C}\,)_X$ - - return the tree rooted at node X $\square$

non-ambiguous grammar $\Rightarrow$ deterministic reconstruction

ANALYSIS AND TREE CONSTRUCTION – string “*a a b b*”







there are three more terminal shifts
(not shown here)

WORST-CASE COMPLEXITY OF BUILDTREE

Since the grammar is supposed to be not ambiguous, the tree size is linear, so the time complexity of *BT* is also cubic (like Earley)

Extending *BT* to ambiguous grammars is possible as well, but to efficiently represent all the trees a Sparse Tree Forest (*STF*) is necessary, which is a graph type more general than a tree (not considered here)

OPTIMIZATION OF THE EARLEY METHOD BY USING LOOK-AHEAD

The items in the sets *E* [i] of the Earley vector *E* can be extended by including look-ahead, computed as it is done in the *ELR* (1) method

By siding a look-ahead to each item, the Earley algorithm avoids to put into the sets *E* [i] items that correspond to choices doomed to failure. This way however, for some grammars the number of items may increase rather than decrease. So the advantage of using look-ahead in Earley is controversial

At present the most efficient implementations of Earley do not use look-ahead