Bottom-up Syntax Analisys
ELR (*k*) Method

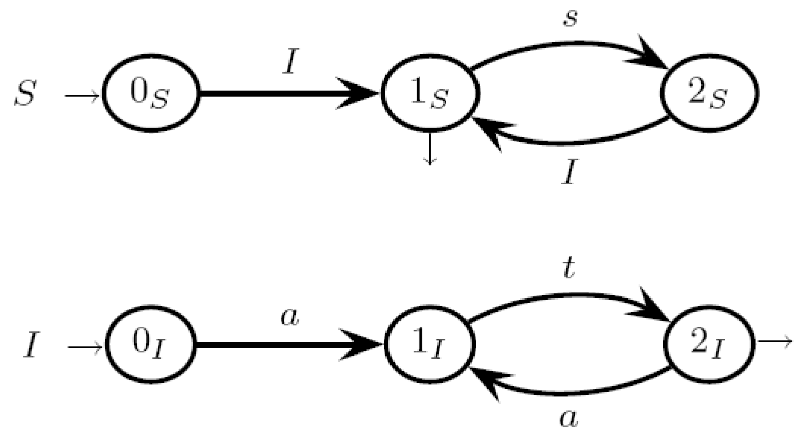
Translated and adapted by L. Breveglieri

EBNF grammar

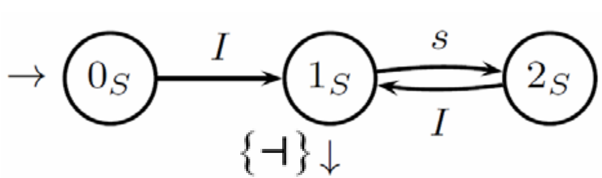
$$S \rightarrow I \ (\ s \ I \ )^*$$

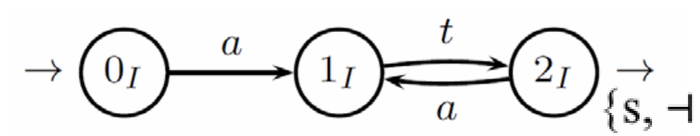
$$I \rightarrow (\ a \ t \ )^+$$

machine net



For the analysis add the *follow sets* of the non-terminals: *follow* (*S*) and *follow* (*I*)
the *follow set* is often called *look-ahead set* or *prospect set*





The follow sets allow us to decide when to

- go on with the analysis and run an automaton transition (*shift* move)
- or recognize a non-terminal and build a sub-tree (*reduce* move)

PARSING TRACE OF STRING $x = a\ t\ a\ t\ s\ a\ t\ \dashv$

Push a *macro-state* (m-state) $\{0_S, 0_I\}$, which contains the initial states of M_S and M_I

The analysis of axiom S starts from I: a kind of ε-move called *closure* or *completion*

stack bottom	string to be parsed (with end-marker) and stack contents								effect after	#
	1	2	3	4	5	6	7			
	a	t	a	t	s	a	t	$\dashv$		
$\{0_S\ 0_I\}$									initialization of the stack	1

current char *cc*

Since from state 0_I there is an a-arc and $cc = a$, make a *shift*

stack bottom	1	2	3	4	5	6	7		effect after	#
	a	t	a	t	s	a	t	$\dashv$		
$\{0_S\ 0_I\}$	a $\{1_I\}$								terminal shift on a: $0_I \xrightarrow{a} 1_I$	2

push *a* to remember the *a* read in the input

stack top

new current char *cc*

Since from state 1_I there is a t-arc and $cc = t$, make a *shift*

stack bottom	1	2	3	4	5	6	7		effect after	#
	a	t	a	t	s	a	t	$\dashv$		
$\{0_S\ 0_I\}$	a $\{1_I\}$	t $\{2_I\}$							terminal shift on t: $1_I \xrightarrow{t} 2_I$	3

from before

	a		t		a	t	s	a	t	$\dashv$		
$\{0_S\ 0_I\}$	a	$\{1_I\}$	t	$\{2_I\}$							terminal shift on t: $1_I \xrightarrow{t} 2_I$	3

Choice between *shift* and *reduce*: since it is *cc* = *a* $\notin$ *follow* (*I*), make a *shift*

	a		t		a		t	s	a	t	$\dashv$	(do not reduce)	
$\{0_S\ 0_I\}$	a	$\{1_I\}$	t	$\{2_I\}$	a	$\{1_I\}$						terminal shift on a: $2_I \xrightarrow{a} 1_I$	4

Since from state 1_I there is a *t*-arc and *cc* = *t*, make a *shift*

	a		t		a		t		s	a	t	$\dashv$		
$\{0_S\ 0_I\}$	a	$\{1_I\}$	t	$\{2_I\}$	a	$\{1_I\}$	t	$\{2_I\}$					terminal shift on t: $1_I \xrightarrow{t} 2_I$	5

from before

	a		t		a		t		s	a	t	$\dashv$		5
$\{0_S\ 0_I\}$	a	$\{1_I\}$	t	$\{2_I\}$	a	$\{1_I\}$	t	$\{2_I\}$					terminal shift on t: $1_I \xrightarrow{t} 2_I$	

Choice between *shift* and *reduce*: since it is *cc* = *s* $\in$ *follow* (*I*), make a *reduce*

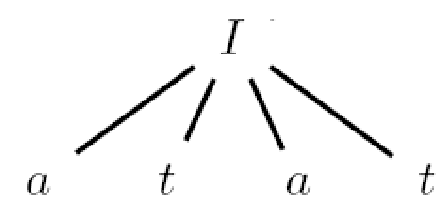
To find the reduction handle, explore the stack back to the initial state 0_I of M_I and pop all the explored portion (except 0_I) – easy strategy to be refined later

$$\{\,0_S\ 0_I\,\}\ \overbrace{a\ \{\,1_I\,\}\ t\ \{\,2_I\,\}\ a\ \{\,1_I\,\}\ t\ \{\,2_I\,\}}^{\textit{segment (handle) to be reduced by: } a\,t\,a\,t \rightsquigarrow I}$$

	I	s	a	t	$\dashv$		6
$\{0_S\ 0_I\}$						since s follows I reduce $a\,t\,a\,t \rightsquigarrow I$	

it does not overwrite the input tape (which is read-only) – it helps us remember the reduction operated

built a fragment of syntax tree



from before

stack						comment	step
	I	s	a	t	$\dashv$		
$\{0_S\ 0_I\}$						since s follows I reduce $a\,t\,a\,t \rightsquigarrow I$	6

Push the identified reduction *a t a t* → *I* : make a *non-terminal shift*

stack							comment	step
	I		s	a	t	$\dashv$		
$\{0_S\ 0_I\}$	I	$\{1_S\}$					nonterminal shift on I: $0_S \xrightarrow{I} 1_S$	7

Choice between *shift* and *reduce*: since it is *cc* = *s* ∉ *follow* (*S*), make a *shift*

stack								comment	step
	I		s		a	t	$\dashv$	(do not reduce)	
$\{0_S\ 0_I\}$	I	$\{1_S\}$	s	$\{2_S\ 0_I\}$				terminal shift on s: $1_S \xrightarrow{s} 2_S$	8

Then make two more *shifts* (here left uncommented)

stack							t	$\dashv$	comment	step
	I		s		a		t	$\dashv$		
$\{0_S\ 0_I\}$	I	$\{1_S\}$	s	$\{2_S\ 0_I\}$	a	$\{1_I\}$			terminal shift on a: $0_I \xrightarrow{a} 1_I$	9

stack										comment	step
	I		s		a		t		$\dashv$		
$\{0_S\ 0_I\}$	I	$\{1_S\}$	s	$\{2_S\ 0_I\}$	a	$\{1_I\}$	t	$\{2_I\}$		terminal shift on t: $1_I \xrightarrow{t} 2_I$	10

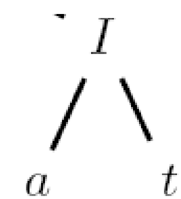
from before

	I		s		a		t		$\dashv$		
$\{0_S\ 0_I\}$	I	$\{1_S\}$	s	$\{2_S\ 0_I\}$	a	$\{1_I\}$	t	$\{2_I\}$		terminal shift on t: $1_I \xrightarrow{t} 2_I$	10

Choice between *shift* and *reduce*: since it is *cc* = $\dashv$ $\in$ *follow* (*S*), make a *reduce*

	I		s		I	$\dashv$		
$\{0_S\ 0_I\}$	I	$\{1_S\}$	s	$\{2_S\ 0_I\}$			since $\dashv$ follows I reduce $a\,t \rightsquigarrow I$	11

built a fragment of syntax tree



Push the identified reduction *a t* → *I* : make a *non-terminal shift*

	I		s		I		$\dashv$		
$\{0_S\ 0_I\}$	I	$\{1_S\}$	s	$\{2_S\ 0_I\}$	I	$\{1_S\}$		nonterminal shift on I: $2_S \xrightarrow{I} 1_S$	12

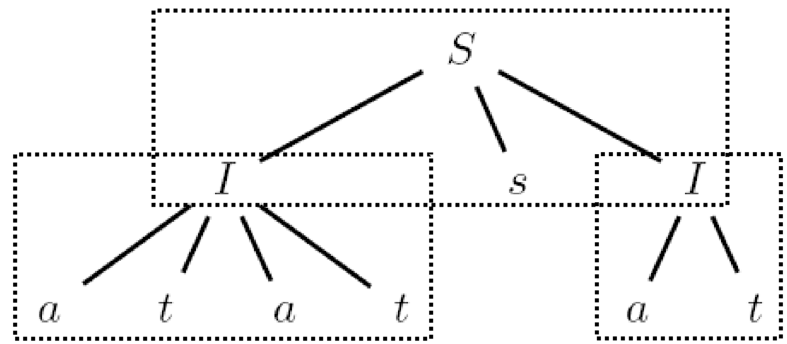
from before

	I		s		I		$\dashv$		
$\{0_S\ 0_I\}$	I	$\{1_S\}$	s	$\{2_S\ 0_I\}$	I	$\{1_S\}$		nonterminal shift on I: $2_S \xrightarrow{I} 1_S$	12

Choice between *shift* and *reduce*: since it is *cc* = $\dashv$ $\in$ *follow* (*I*), make a *reduce*

	S	$\dashv$		
$\{0_S\ 0_I\}$			since $\dashv$ follows I reduce $I\ s\ I \rightsquigarrow S$	13

built the complete syntax tree



previous fragment

previous fragment

Analysis end and acceptance condition

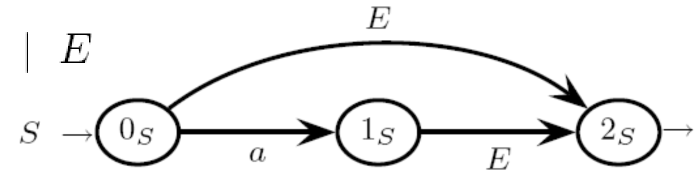
	the input string is reduced to the axiom S	$\dashv$		
$\{0_S\ 0_I\}$	the stack contains only the axiom $\{0_S\ 0_I\}$		stop and accept	14

It is not easy to find the right reduction – it may be necessary to scan the string at a long distance

example with another grammar

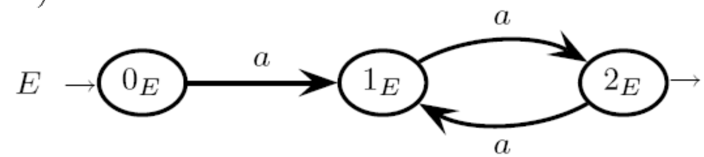
the strings of even length $(aa)^+ = L(0_E)$ and odd a $(aa)^+ = a\,L(1_S)$ are derived in different ways

$S \to a\,E \mid E$



$E \to (a\,a)^+$

net



Derivation identifiable only when meeting the terminator

Analysis of string "*aaa*": in three steps the stack contains

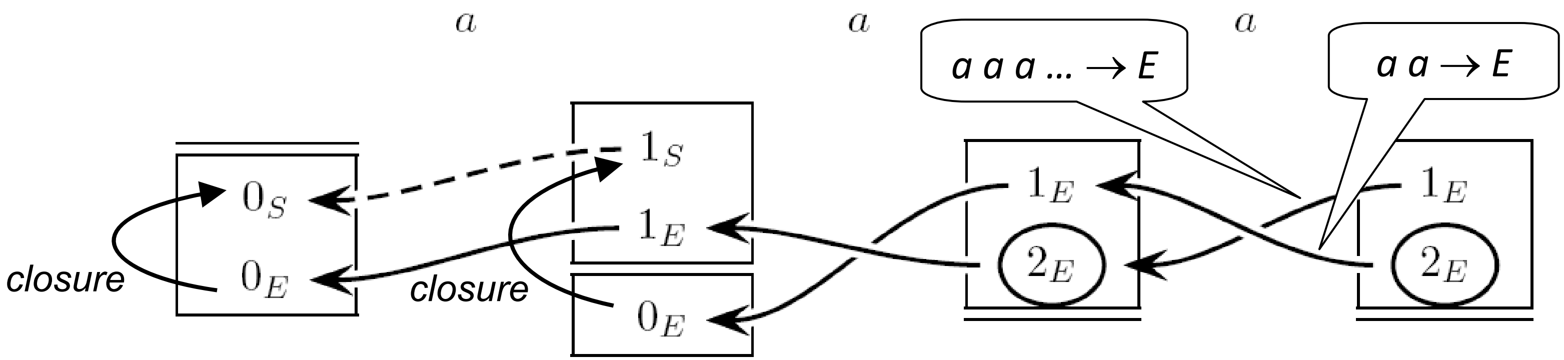
	$\{\,0_S\; 0_E\,\}\; a\; \{\,1_S\; 1_E\; 0_E\,\}\; a\; \{\,2_E\; 1_E\,\}\; a\; \{\,1_E\; 2_E\,\}$

The second m-state $\{1_S, 1_E, 0_E\}$ indicates three computations

$$0_S \xrightarrow{a} 1_S \qquad 0_S \xrightarrow{\varepsilon} 0_E \xrightarrow{a} 1_E \qquad 0_S \xrightarrow{a} 1_S \xrightarrow{\varepsilon} 0_E$$

From which 0_E do we reduce ? from $\{0_S, 0_E\}$, i.e., $aaa \ldots \to E$, or from $\{1_S, 1_E, 0_E\}$, i.e., $aa \to E$?

To decide, we keep the analysis history in the stack by means of backward-directed pointers



Since the pointers can be modeled as bounded integers, the *PDA* still has a finite stack alphabet

SYSTEMATIC CONSTRUCTION OF THE BOTTOM-UP SYNTAX ANALYZER

shortly called *PDA*

- construction of the pilot graph
  - the pilot drives (pilots) the *PDA*
  - in each macro-state (m-state) the pilot incorporates all the information about any possible phrase form that reaches the m-state
    - each m-state contains machine states with look-ahead (prospection)
      - look-ahead: which chars we expect to see in the input at reduction time; it corresponds to the followers *follow* (*A*) we have seen before
- the m-states are used to build a few analysis threads in the stack, which correspond to possible derivations
  - $\equiv$ computations of the machine network
  - $\equiv$ paths with ε-arcs at each machine change, labeled with the scanned string
- verification of determinism conditions on the pilot graph: three problems
  - *shift-reduce* conflict
  - *reduce-reduce* conflict
  - (less frequently) two or more paths that merge into one state with non-disjoint look-ahead: *convergence conflict*
- if the determinism test is passed, the *PDA* can analyze the string deterministically
- the *PDA* uses the information stored in the pilot graph and in the stack

A FEW DEFINITIONS

SET OF INITIALS – similar (but not identical) to Berry-Sethi

Set of chars found starting from state q_A of machine M_A of the net M

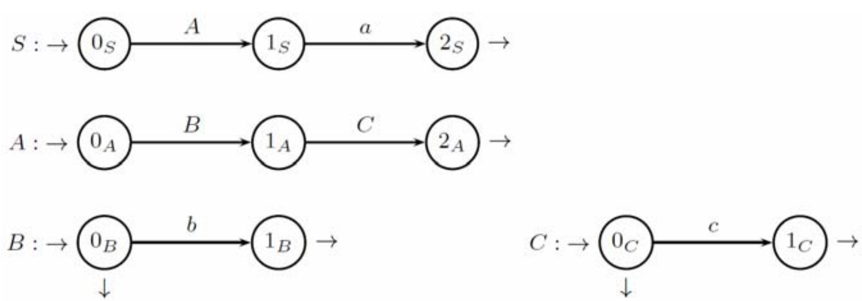
$$Ini(q_A) = Ini\,(L(q_A)) = \{a \in \Sigma \mid a\Sigma^* \cap L(q_A) \neq \emptyset\}$$

Defined in the three cases aside *by logical clauses*

1. $a \in Ini(q_A)$ if $\exists$ arc $q_A \xrightarrow{a} r_A$
2. $a \in Ini(q_A)$ if $\exists$ arc $q_A \xrightarrow{B} r_A$ $\wedge\ a \in Ini(0_B)$ (where 0_B is the initial state of machine M_B)
3. $a \in Ini(q_A)$ if $\exists$ arc $q_A \xrightarrow{B} r_A$ $\wedge\ L(0_B)$ is nullable $\wedge\ a \in Ini(r_A)$

EXAMPLE

$S \to Aa \qquad A \to BC \qquad B \to b \mid \varepsilon \qquad C \to c \mid \varepsilon$



$$Ini(0_S) = Ini(0_A) \cup \overbrace{Ini(1_S)}^{\{a\}} \text{ because } L(0_A) \text{ is nullable}$$

$$Ini(0_A) = \overbrace{Ini(0_B)}^{\{b\}} \cup Ini(1_A) \text{ because } L(0_B) \text{ is nullable}$$

$$Ini(1_A) = \overbrace{Ini(0_C)}^{\{c\}} \cup \overbrace{Ini(2_A)}^{\emptyset} \text{ because } L(0_C) \text{ is nullable}$$

$$Ini(0_S) = \{b\} \cup \{c\} \cup \{a\}$$

ITEM (or *candidate*) $\langle q_B, a \rangle$ in $Q \times (\Sigma \cup \{ \dashv \})$

Two or more items with the same state can be grouped into one item

$$\{ \langle q, a_1 \rangle, \langle q, a_2 \rangle, \ldots \langle q, a_k \rangle \} \quad \Rightarrow \quad \langle q, \{ a_1, a_2, \ldots a_k \} \rangle$$

An item with a machine final state is said to be a *reduction item*

CLOSURE

Function *closure* (*C*) computes a kind of closure of a set *C* of items with look-ahead

Repeatedly apply this (recursive) clause until a fixed point is reached

$closure(C) = C$ initial setting

recursive step

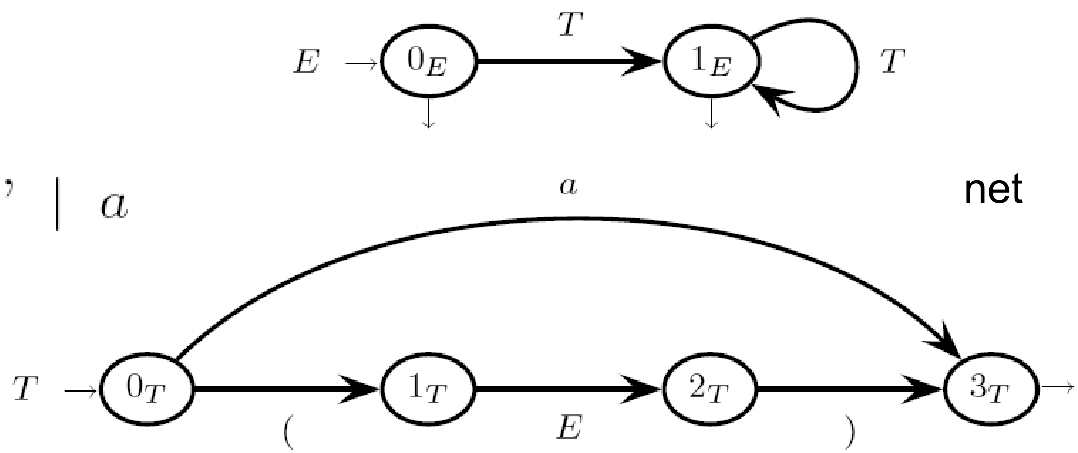
$$\langle 0_B, b \rangle \in closure(C) \quad \text{if} \begin{cases} \exists \text{ candidate } \langle q, a \rangle \in C & \text{and} \\ \exists \text{ arc } q \xrightarrow{B} r \text{ in } \mathcal{M} & \text{and} \\ b \in Ini(\, L(r) \cdot a \,) \end{cases}$$

"*a*" comes in when
L (*r*) is nullable

RUNNING EXAMPLE – an *EBNF* grammar

$\Sigma = \{\ a,\ `(',\ `)'\ \} \quad V = \{\ E,\ T\ \}$ axiom *E*

$$P = \begin{cases} E \to T^* \\ T \to `('\ E\ `)' \ \mid\ a \end{cases}$$



Sample strings

$$\varepsilon \quad a\,a \quad (\ ) \quad (\ a\ ) \quad (\ )\ a \quad (\ a\,a\ ) \quad (\ (\ a\ )\ ) \quad (\ (\ )\ a\ a\ )$$

Each machine state p_A of a m-state *I* has a look-ahead (or prospection) set $\pi \subseteq \Sigma \cup \{\ \dashv\ \}$

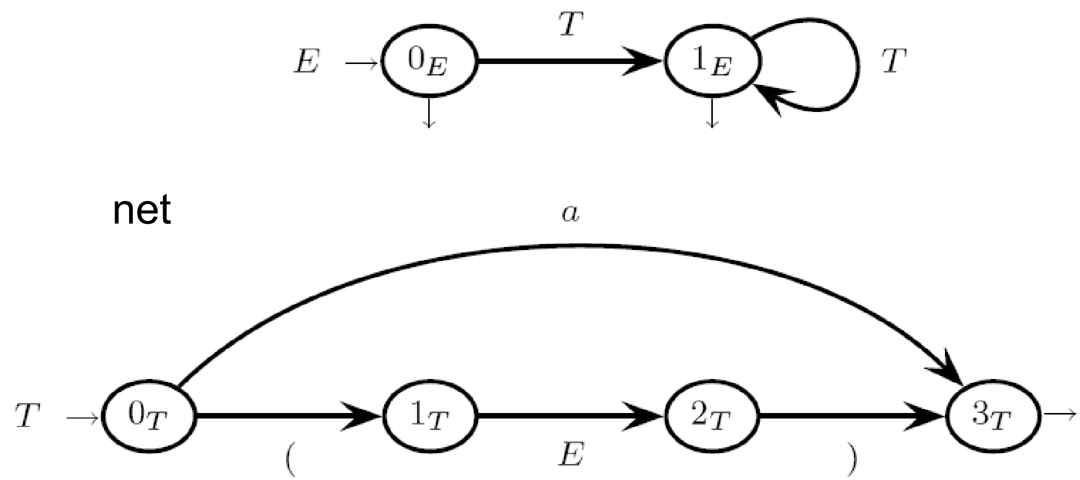
Item (or candidate) $\langle p_A, \pi \rangle \in I$ contains the chars that may follow a string generated by *A*

When the analysis reaches the end of a string generated by a machine M_A,
the *PDA* makes a reduction with current char $cc \in$ look-ahead, i.e., $cc \in \pi$

CLOSURE EXAMPLE – for the running example

$$\Sigma = \{\ a,\ `(\text{'},\ `)\text{'}\ \} \qquad V = \{\ E,\ T\ \} \qquad P = \begin{cases} E \to T^* \\ T \to `(\text{'}\ E\ `)\text{'}\ \mid\ a \end{cases}$$

net



C	function closure(C)		
$\langle 0_E, \dashv \rangle$	$\langle 0_E, \dashv \rangle$	$\langle 0_T, \{\ a,\ `(\text{'},\ \dashv\ \} \rangle$	
$\langle 1_T, \dashv \rangle$	$\langle 1_T, \dashv \rangle$	$\langle 0_E, `)\text{'} \rangle$	$\langle 0_T, \{\ a,\ `(\text{'},\ `)\text{'}\ \} \rangle$

SHIFT OPERATION – also denoted θ

We are close to defining an algorithm for constructing the pilot graph

Define the *shift* operation θ

with a *shift* operation the item look-ahead does not change – a new look-ahead is created by *closure*

$$\begin{cases} \vartheta\ (\langle p_A,\ \rho \rangle\ ,\ X\ ) = \langle\ q_A,\ \rho\ \rangle & \text{if the arc } p_A \xrightarrow{X} q_A \text{ exists} \\ \text{the empty set} & \text{otherwise} \end{cases}$$

A *shift* corresponds to a transition in a machine *Y*

- if $X = c$ is a terminal symbol, then *shift* is a *PDA* move that reads a char *c* in the input
- if *X* is a non-terminal symbol, then *shift* is a *PDA* ε-move after a reduction $z \rightarrow X$ and it does not read any input
- machine *Y* runs a transition with non-terminal label *X*
- the analysis goes on after recognizing an input substring $z \in L\ (X)$ derivable from the non-terminal *X*

The shift operation extends to sets of items (i.e., *m-states*) as obvious

$$\vartheta\ (\ C,\ X\ ) = \bigcup_{\forall \text{ candidate } \gamma \in C} \vartheta\ (\gamma, X)$$

HOW TO BUILD THE PILOT GRAPH

A *macro-state* (shortly *m-state*) is defined as a set of items

The pilot is a DFA, named $\mathcal{P}$, defined by the following entities:

- the set R of m-states
- the *pilot alphabet* is the union $\Sigma \cup V$ of the terminal and nonterminal alphabets, to be also named the *grammar symbols*
- the initial m-state, I_0, is the set: $I_0 = closure(\langle 0_S, \dashv \rangle)$
- the m-state set $R = \{ I_0, I_1, \ldots \}$ and the state-transition function $\vartheta \colon R \times (\Sigma \cup V) \to R$ are computed starting from I_0

OBSERVATIONS AND TERMINOLOGY

Incremental construction: it ends when nothing changes any longer

No final m-state: the pilot does not recognize strings, it controls the *PDA*

The items in each m-state *I* of the pilot are parted into two groups

base, contains the items obtained after a shift $\Rightarrow$ non-initial states

closure, contains the items obtained after a closure $\Rightarrow$ initial states

Call *kernel* of a m-state *I* the set of the m-states of *I* without look-ahead

PILOT GRAPH CONSTRUCTION ALGORITHM

```
R' := { I_0 }                                  - - prepare the initial m-state I_0
- - loop that updates the m-state set and the state-transition function
do
    R := R'                                    - - update the m-state set R
    - - loop that computes possibly new m-states and arcs
    for ( each m-state I ∈ R and symbol X ∈ Σ ∪ V ) do
        - - compute the base of a m-state and its closure
        I' := closure (ϑ (I, X))
        - - check if the m-state is not empty and add the arc
        if ( I' ≠ ∅ ) then
            add arc I --X--> I' to the graph of ϑ
            - - check if the m-state is a new one and add it
            if ( I' ∉ R ) then
                add m-state I' to the set R'
            end if
        end if
    end for
while ( R ≠ R' )                               - - repeat if the graph has grown
```

RUNNING EXAMPLE – FIRST THREE M-STATES

I_0 = *closure* $(\langle 0_E, \{\dashv\}\rangle) = \{\ \langle 0_E, \{\dashv\}\rangle, \langle 0_T, \{a, (, \{\dashv\}\}\rangle\ \}$

for the item $\langle 0_E, \{\dashv\}\rangle$ of I_0 and the T-arc $0_E \rightarrow 1_E$ do

compute *shift* $(I_0, T) = \{\ \langle 1_E, \{\dashv\}\rangle\ \}$

compute *closure* $(\langle 1_E, \{\dashv\}\rangle) = \{\ \langle 1_E, \{\dashv\}\rangle, \langle 0_T, \{a, (, \dashv\}\rangle\ \}$

add to the pilot a new m-state I_1 and a new T-arc $I_0 \rightarrow I_1$

for the item $\langle 0_T, \{a, (, \dashv\}\rangle$ of I_0 and the a-arc $0_T \rightarrow 3_T$ do

compute *shift* $(I_0, a) = \{\ \langle 3_T, \{a, (, \dashv\}\rangle\ \}$

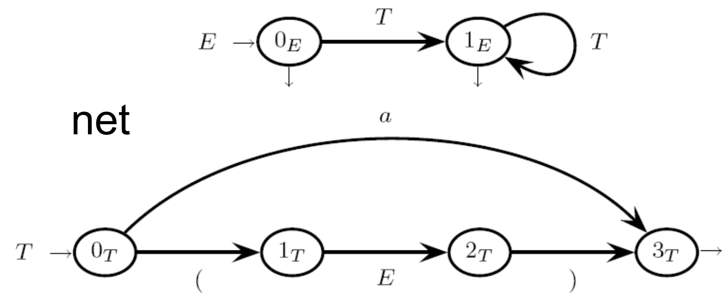
compute *closure* $(\langle 3_T, \{a, (, \dashv\}\rangle) = \{\ \langle 3_T, \{a, (, \dashv\}\rangle\ \}$

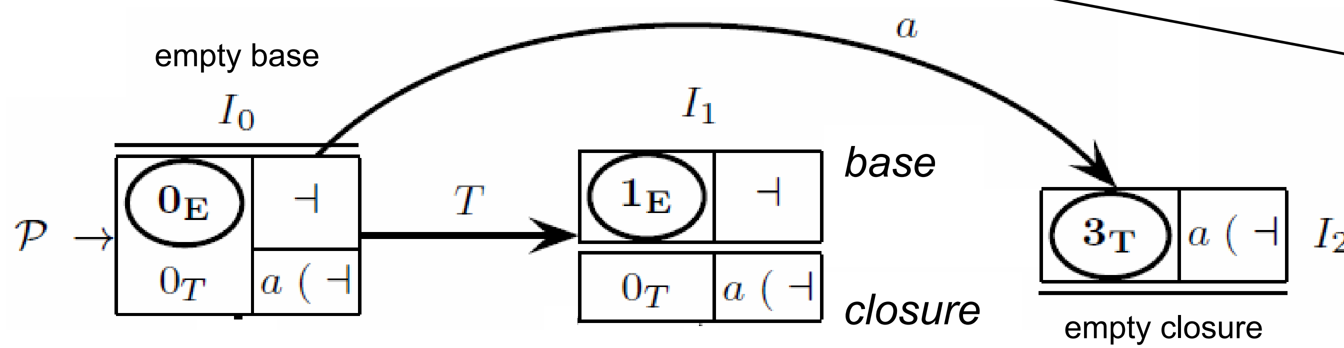
add to the pilot a new m-state I_2 and a new a-arc $I_0 \rightarrow I_2$

$$E \rightarrow T^*$$

$$T \rightarrow \text{'('}\ E\ \text{')'}\ \mid\ a$$

net





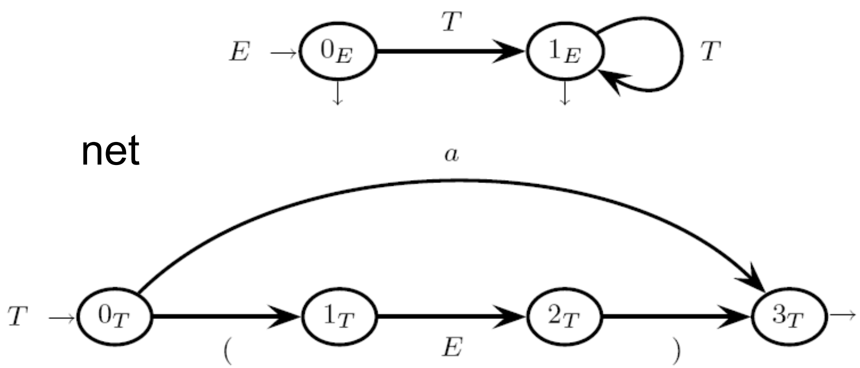
The machine final states in the pilot m-states are highlighted and encircled

The initial m-state I_0 only consists of a closure – its base is empty by construction

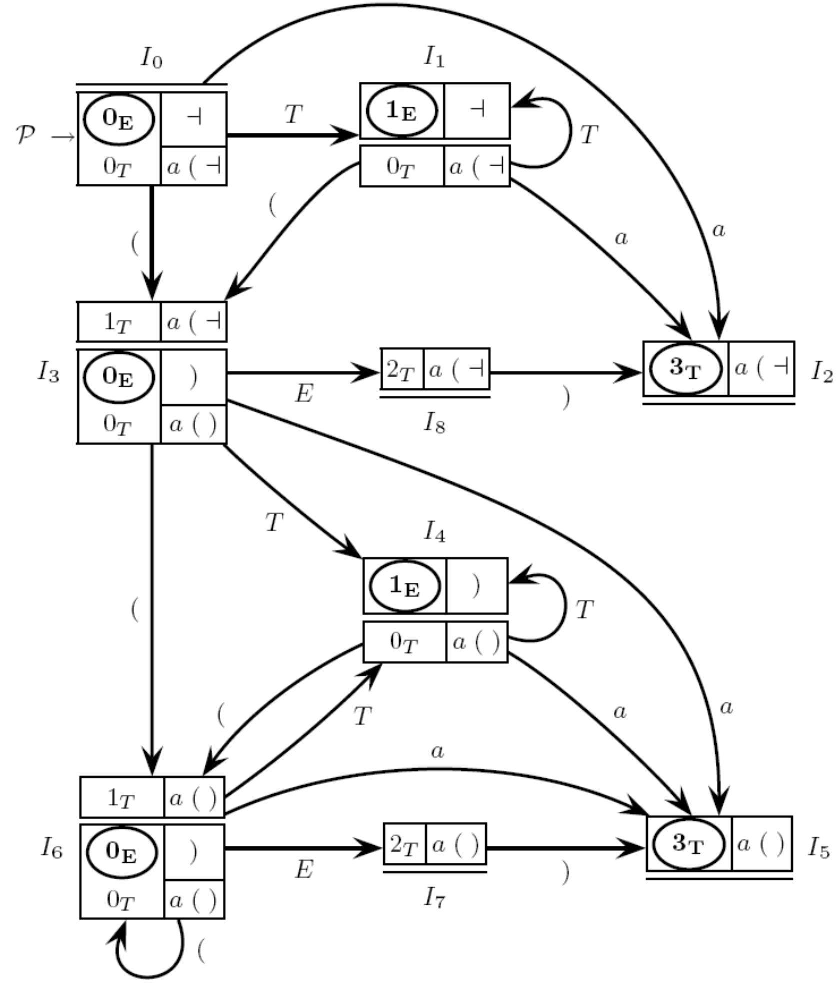
Base and closure are graphically divided by a double line – the closure of I_2 is empty

A new look-ahead is created by a closure – a shift just inherits the previous look-ahead

COMPLETE PILOT GRAPH



net



pilot graph

State pairs with the same kernel (kernel-equivalent):

(I_3, I_6) (I_1, I_4) (I_2, I_5) (I_7, I_8)

If an m-state contains an item with a final state, then the *PDA* makes a reduction move

The look-ahead of the reduction item indicates the input characters expected at reduction time

The *PDA* verifies the current input char *cc*

- if *cc* $\in$ look-ahead, *PDA* makes a reduction
- else (if possible) *PDA* reads input char *cc* and makes a shift on an arc with label *cc*
- otherwise *PDA* stops and rejects

CONDITION *ELR* (1) – makes determinism possible

PART 1 – no m-state has any

shift-reduce conflict

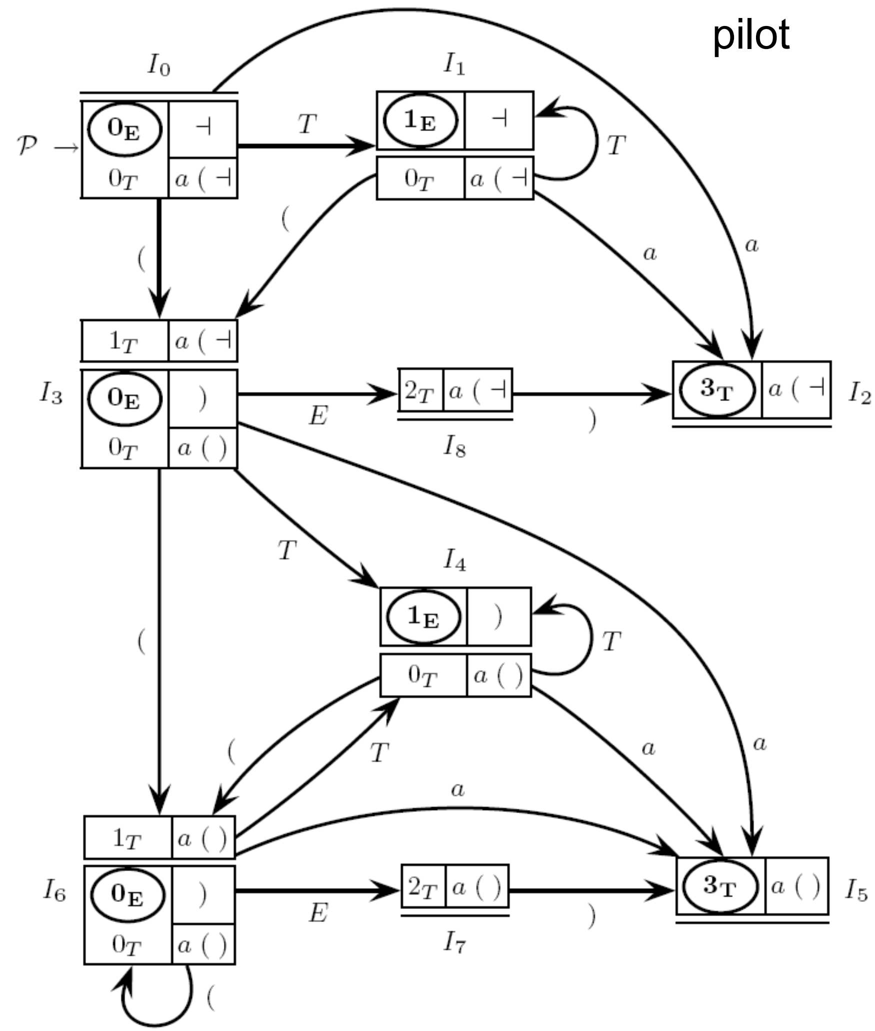
$\exists$ a reduction item with look-ahead that overlaps with the terminal symbols on the outgoing arcs
$\Rightarrow$ *PDA* is unable to choose shift or reduction

reduce-reduce conflict

$\exists$ two or more reduction items with look-ahead that overlap $\Rightarrow$ *PDA* is unable to choose which reduction

m-state	*reduce-shift conflict*	*reduce-reduce conflict*
I_0	No: the reduction candidate is $\langle 0_E, \dashv \rangle$; the characters scanned are a and (, disjoint from $\{\dashv\}$	No: only one reduction candidate
I_1	No: the reduction candidate is $\langle 1_E, \dashv \rangle$; the characters scanned are a and (	No: only one reduction candidate
I_2	No: only one candidate which is a pure reduction	No: only one reduction candidate
I_3	No: the reduction candidate is $\langle 0_E,) \rangle$; the characters scanned are a and (	No: only one reduction candidate
I_4	No: the reduction candidate is $\langle 1_E,) \rangle$; the characters scanned are a and (	No: only one reduction candidate
I_5	No: only one candidate which is a pure reduction	No: only one reduction candidate
I_6	No: the reduction candidate is $\langle 0_E,) \rangle$; the characters scanned are a and (	No: only one reduction candidate
I_7	No: pure shift m-state	No: pure shift m-state
I_8	No: pure shift m-state	No: pure shift m-state

pilot



CONDITION *ELR* (1) – PART 2 – no transition has any *convergence conflict*

A m-state has a *multiple transition* if it contains two (or more) items $\langle\, p,\ \pi \,\rangle$ and $\langle\, r,\ \rho \,\rangle$ such that their two (or more) next states $\delta(p, X)$ and $\delta(r, X)$ are defined for a symbol X (terminal or non)

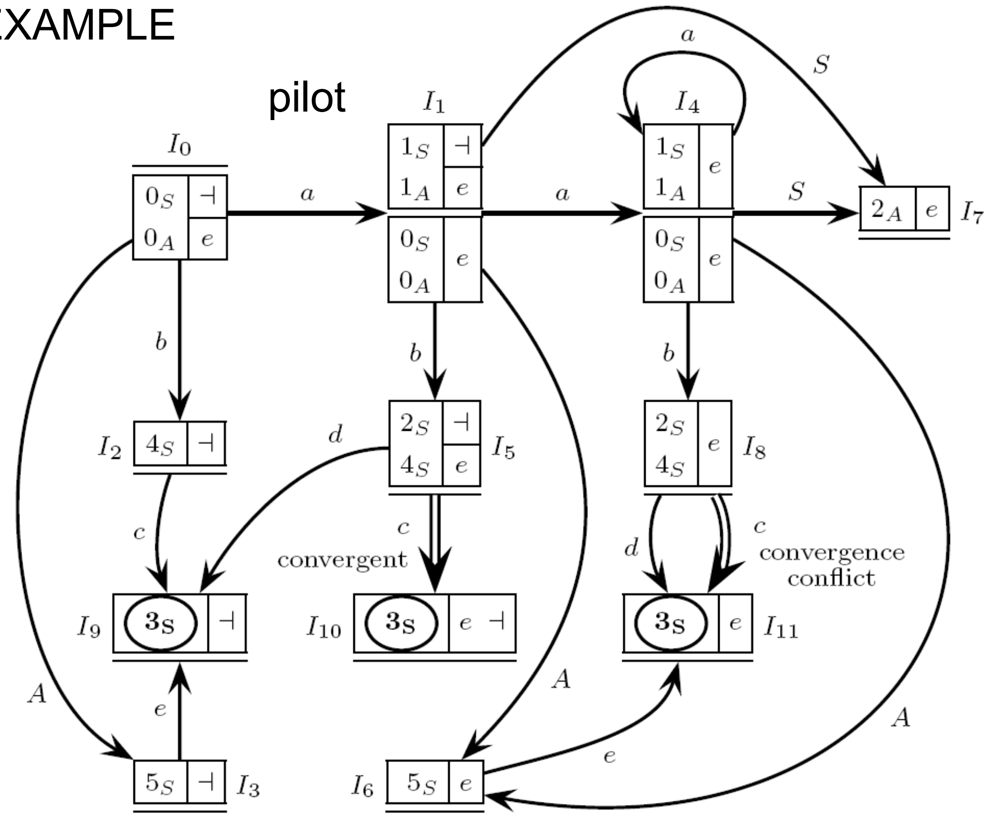
A multiple transition (see before) is *convergent* if $\quad \delta(p, X) = \delta(r, X)$

A (multiple) convergent transition has a *conflict* if $\quad \pi \cap \rho \neq \emptyset$

Two paths merge into one state and the *PDA* is unable to choose one reduction

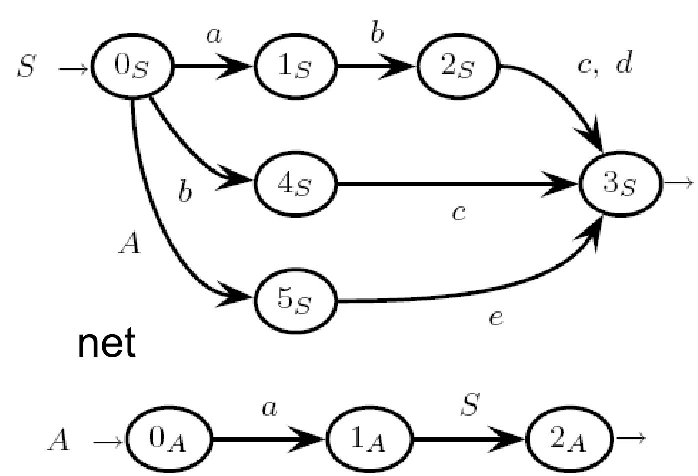
EXAMPLE

pilot



$$S \rightarrow a\, b\, (\, c \mid d \,) \mid b\, c \mid A\, e$$

$$A \rightarrow a\, S$$

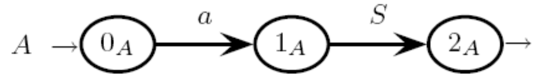


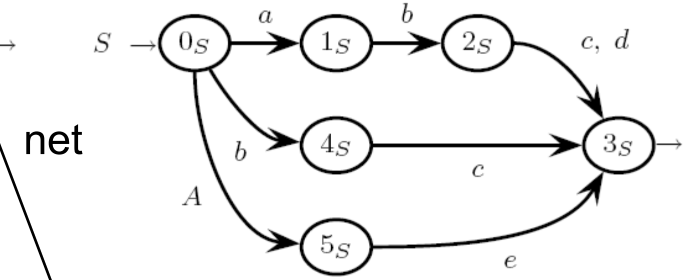
net

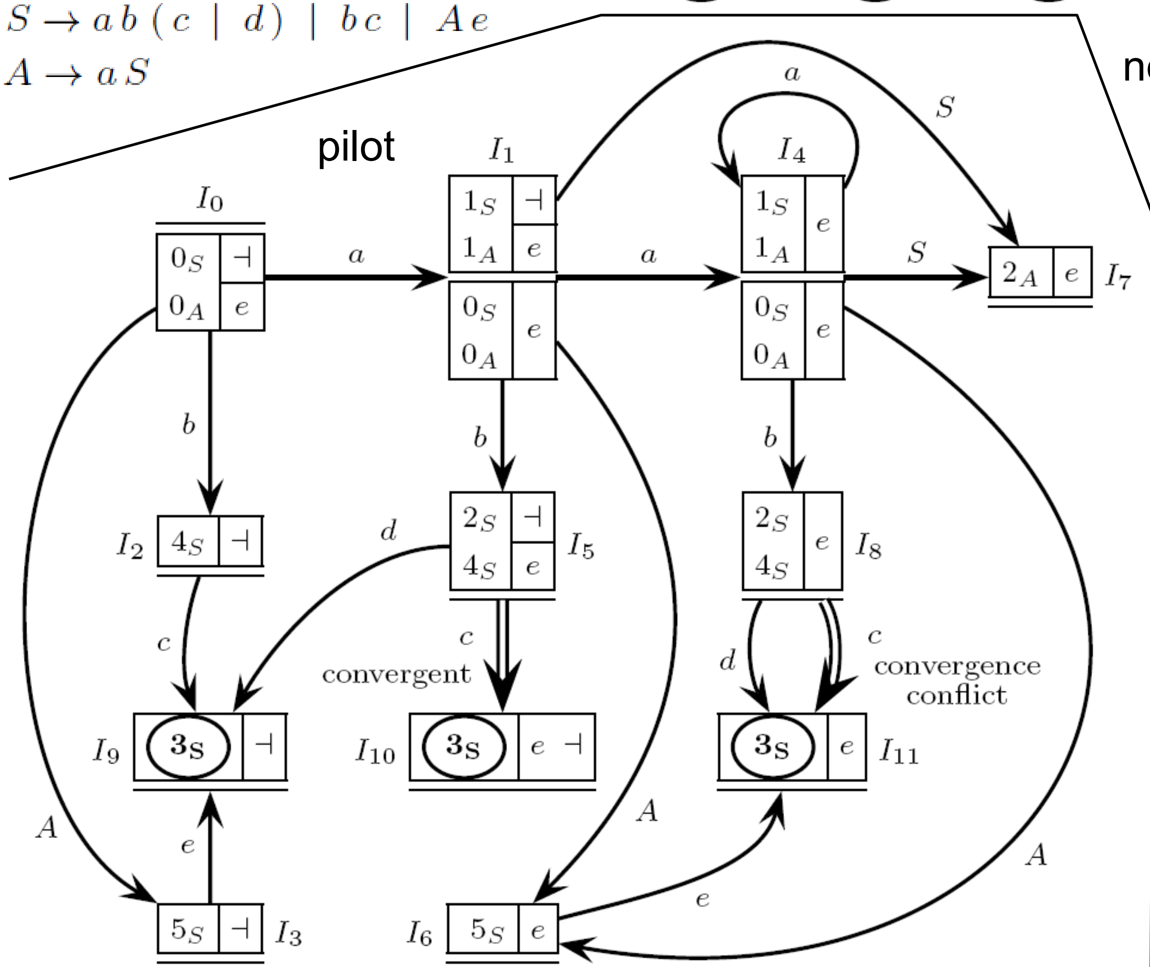
EXAMPLE – convergence

$S \to a\,b\,(c \mid d) \mid b\,c \mid A\,e$

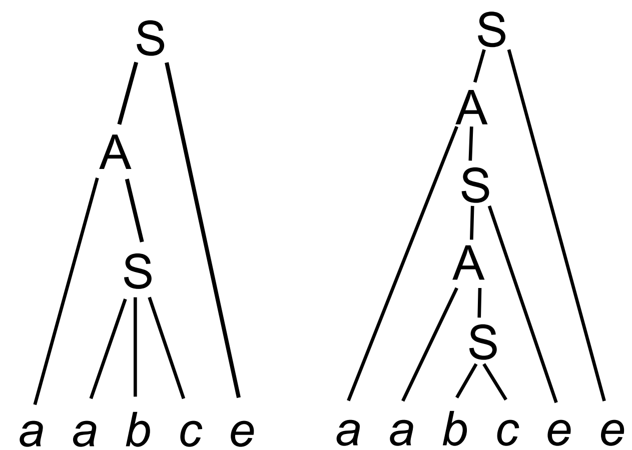
$A \to a\,S$







when analyzing string "*aabce*", after reading char *c* the *PDA* is uncertain between reductions *b c* → *S* and *a b c* → *S*



There are two net computations that can generate the string "*a a b c*", and both ones go into the same m-state with the same look-ahead

$$0_S \xrightarrow{\varepsilon} 0_A \xrightarrow{a} 1_A \xrightarrow{\varepsilon} 0_S \xrightarrow{a} 1_S \xrightarrow{b} 2_S \xrightarrow{c} 3_S \qquad 0_S \xrightarrow{\varepsilon} 0_A \xrightarrow{a} 1_A \xrightarrow{\varepsilon} 0_S \xrightarrow{\varepsilon} 0_A \xrightarrow{a} 1_A \xrightarrow{\varepsilon} 0_S \xrightarrow{b} 4_S \xrightarrow{c} 3_S$$

HOW THE PDA WORKS UNDER THE CONTROL OF THE PILOT GRAPH

The *PDA* scans a string and executes a sequence of shift and reduction moves

The *PDA* pushes groups of items and starts from those in the initial pilot m-state

Each m-state item becomes a 3-tuple by adding to it a backward-directed pointer that helps to reconstruct the different analysis threads constructed in parallel

The *PDA* decides whether to scan or reduce basing on the look-ahead in the pilot

If the condition *ELR* (1) is satisfied (all parts), then the *PDA* is deterministic

CRUCIAL ASPECT – LENGTH OF THE REDUCTION HANDLE TO BE POPPED

The length is not fixed, as a rule right part may contain regular operators like $*$ or $+$
$\Rightarrow$ a rule may generate phrases of unbounded length

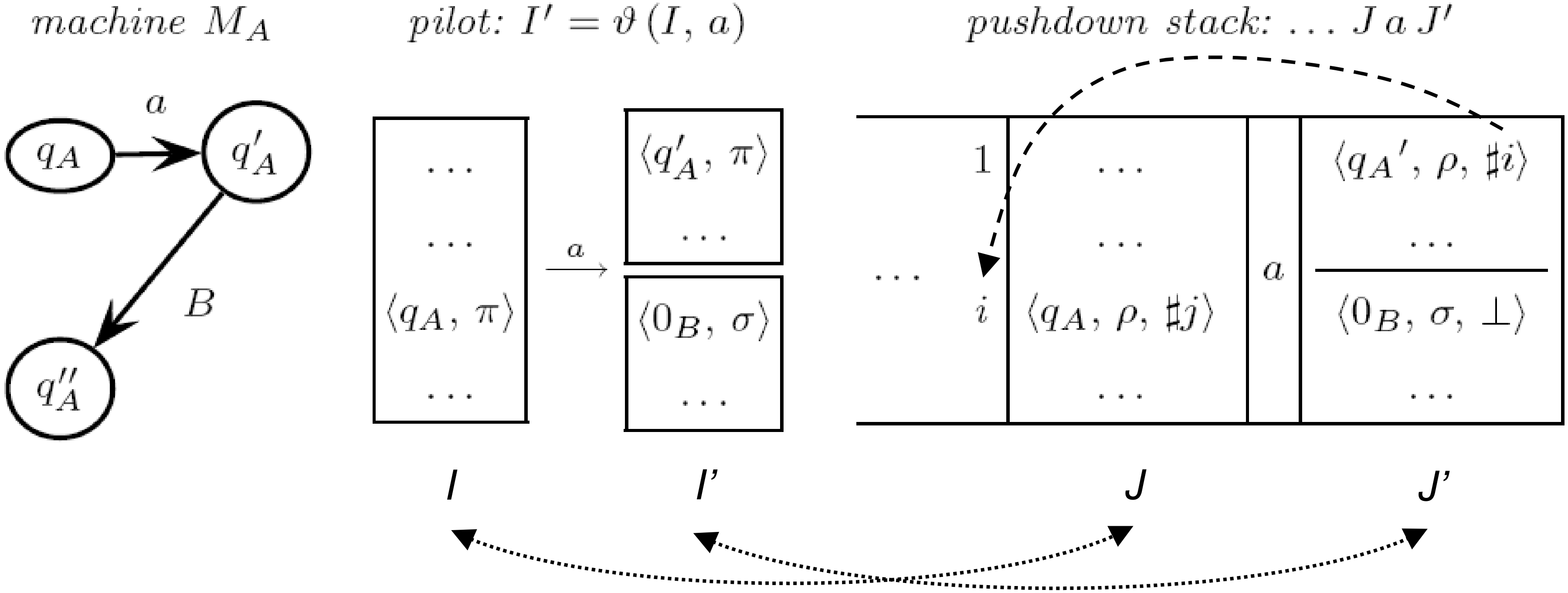
The reduction handle length is determined at reduction time by using the pointers

The pointer chain is followed backwards unto where the analysis thread started

A pointer value “$\perp$” identifies a thread start point and all the closure items have a pointer value “$\perp$”, so these items are the start points of new threads

A 3-tuple with pointer $\neq \perp$ continues an already started thread; the pointers $\neq \perp$ are displayed as $\#i$; *a* pointer value $\#i$ means that the pointer is targeted to the *i*-th item (from top to bottom) in the previous stack element

EXPLANATION OF THE SHIFT MOVE – a correspondence at three levels

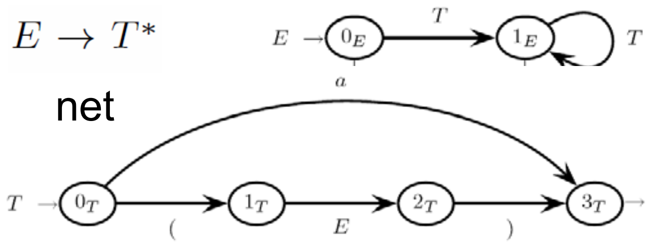


The stack m-states *J*, *J'* are similar to the corresponding pilot m-states *I*, *I'*, but their items have a backward-directed pointer

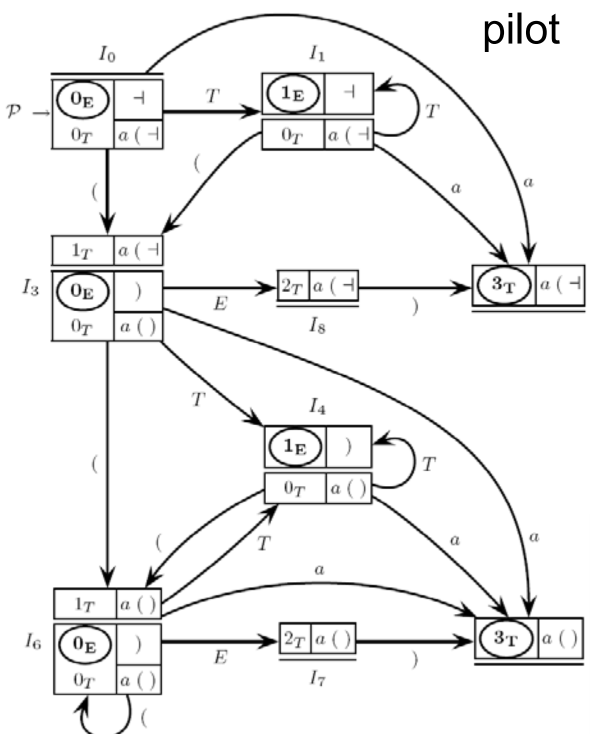
NOTE – the two look-aheads σ and ρ may be different notwithstanding they appear in two items with the same state, because the stack item in *J* may result from splitting the pilot item in *I* (this event may happen only if the transition is convergent – later an example)

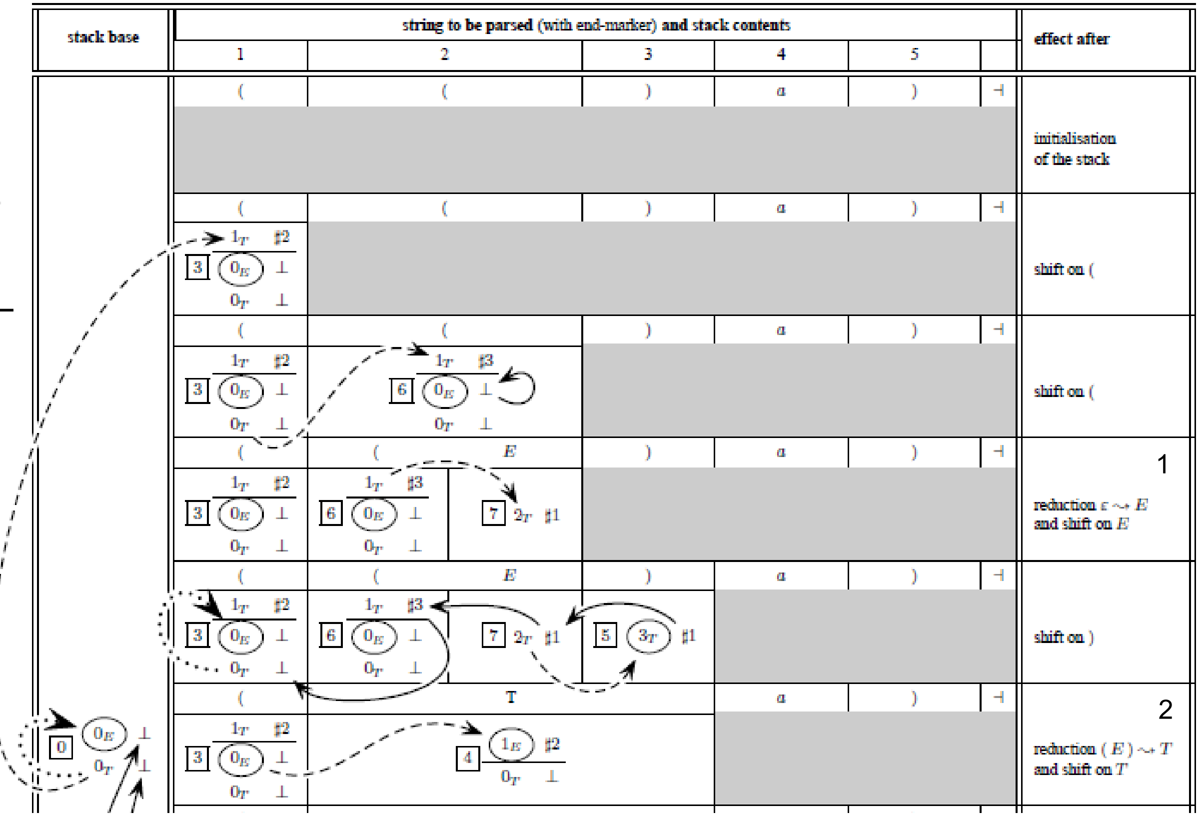
For legibility we do not show the pushed input symbols and the item look-ahead symbols

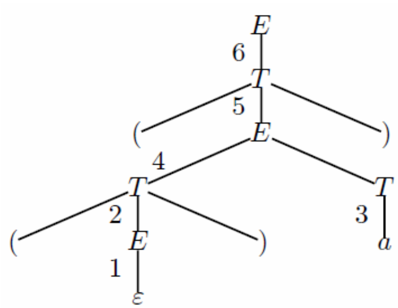
$E \to T^*$



$T \to$ ‘(’ E ‘)’ | a







tree with reductions

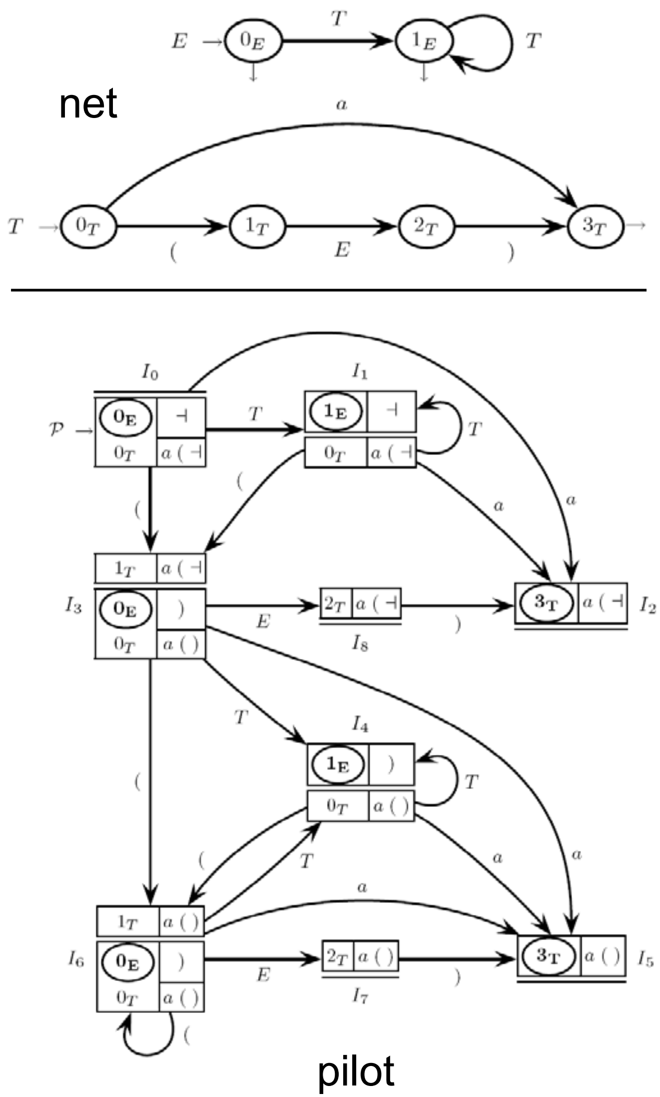
MEANING OF THE ARROWS

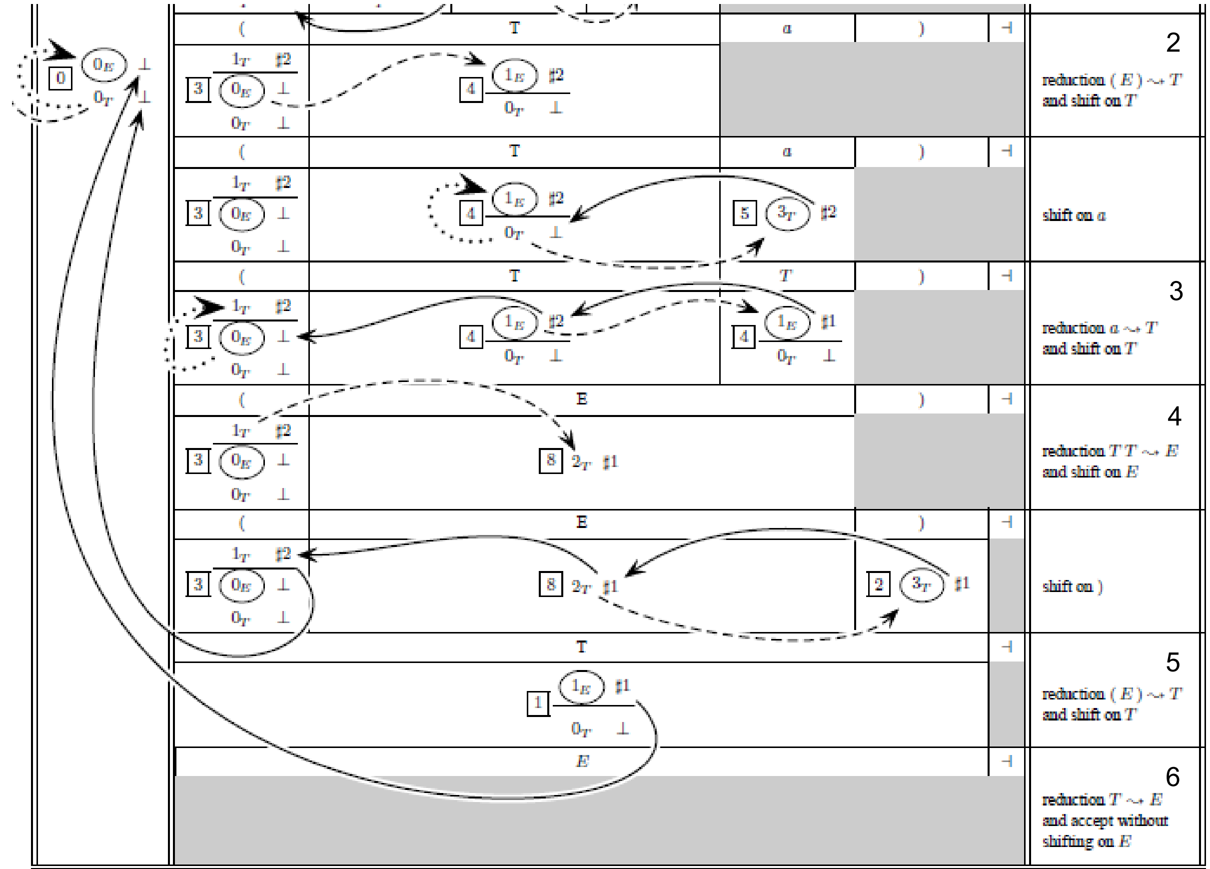
shift

reduction (*handle*)

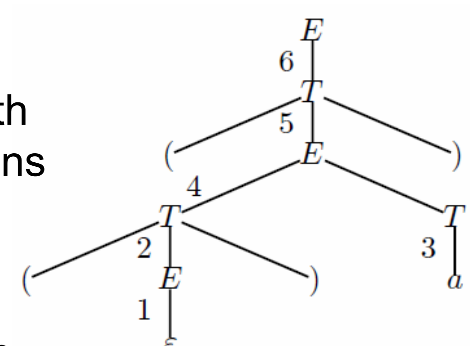
closure

$\boxed{k}$ identifies the m-state

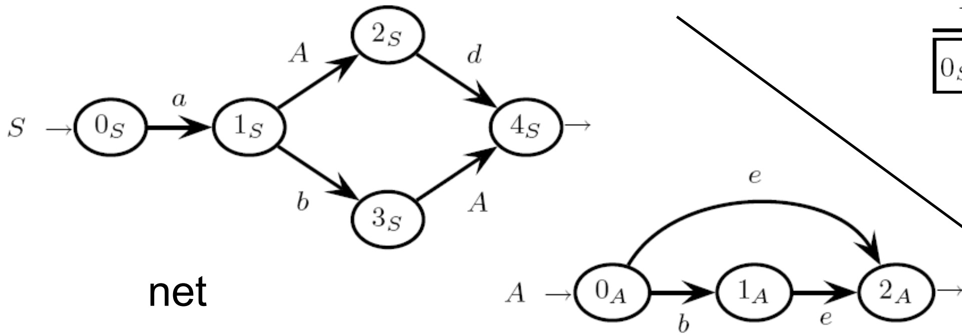


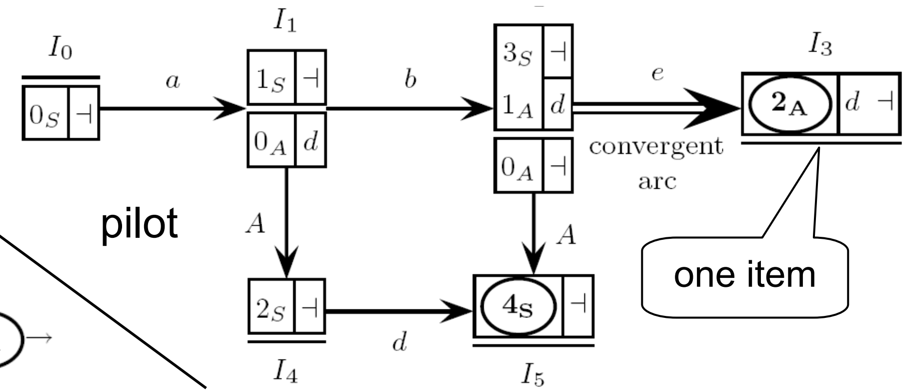


tree with reductions

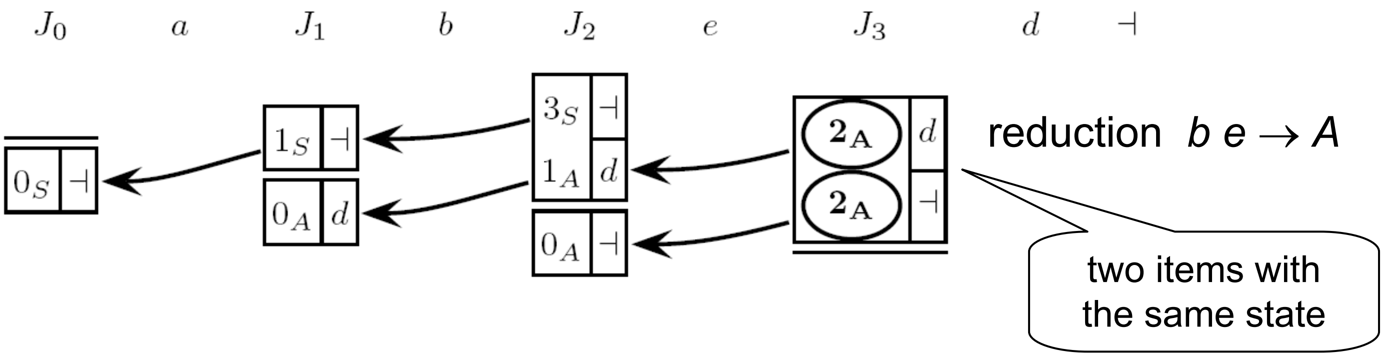


EXAMPLE – if the pilot has a convergent transition, in the *PDA* stack there may be an element that has two or more items with the same state

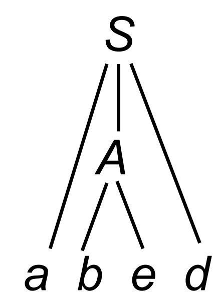


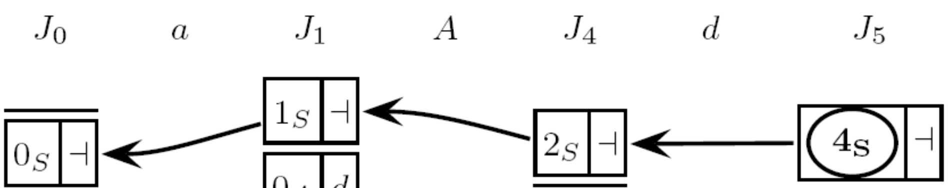


The analysis of the string "*a b e d* ⊣ " leads to J_3 with item in the same state 2_A; the pointers are not displayed as *#i*, but as arrows; to choose the correct reduction, use the look-ahead



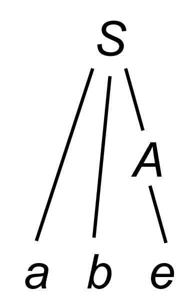
reduction $b\ e \rightarrow A$





reduction $a\ A\ d \rightarrow S$

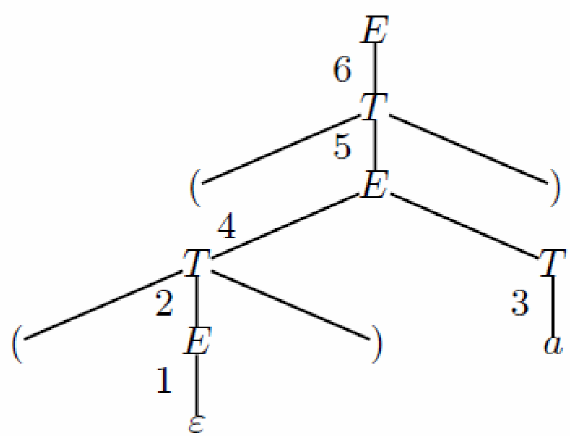
accepts string "*a b e* ⊣ "

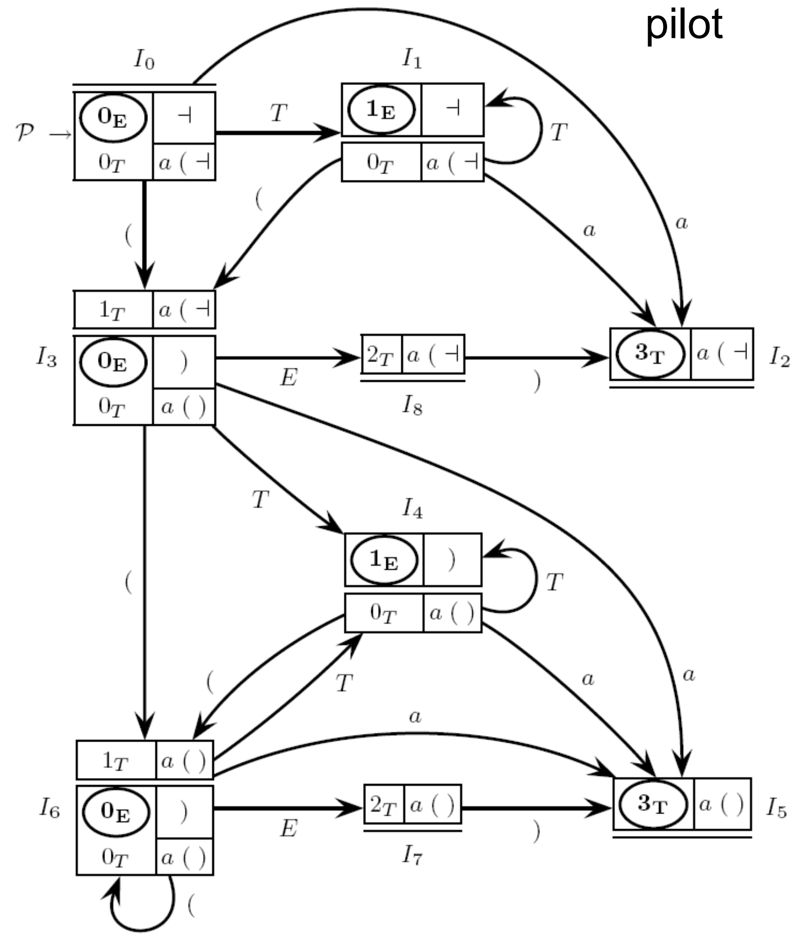


OBSERVATION – what does the pilot recognize ? (if all the m-states are final)

It recognizes all the strings that come from scan or reduction and appear in the stack

Such strings are called *viable prefixes* (or *handle prefixes*)





For the sample string “ (( ) a) ”
the viable prefixes are

(, ((, ((*E*, ((*E*), (*T*, (*Ta*, (*TT*, (*E*, (*E*), *T*

COMPUTATIONAL COMPLEXITY OF THE ANALYSIS

When analyzing a string x of length $n = |x|$, the number of elements in the stack is $\leq n + 1$

To count the number of PDA moves, consider these contributes

$n_T =$ # terminal shifts $\quad$ $n_N =$ # non-terminal shifts $\quad$ $n_R =$ # reductions

Obviously it holds $n_T = n$ and $n_N = n_R$, as for each reduction there is one non-terminal shift

$\Rightarrow$ the total number of *PDA* moves is

$$n_T + n_N + n_R = n + 2 \times n_R$$

Furthermore

- number of reductions with one or more terminals, e.g., $A \to a$, $A \to a\ B$, $A \to B\ a\ C$, is $\leq n$
- number of reductions of type $A \to \varepsilon$, i.e., null, or $A \to B$, i.e., copy, is linearly bounded by n
- number of reductions without any terminals, e.g., $A \to B\ C$, is linearly bounded by n

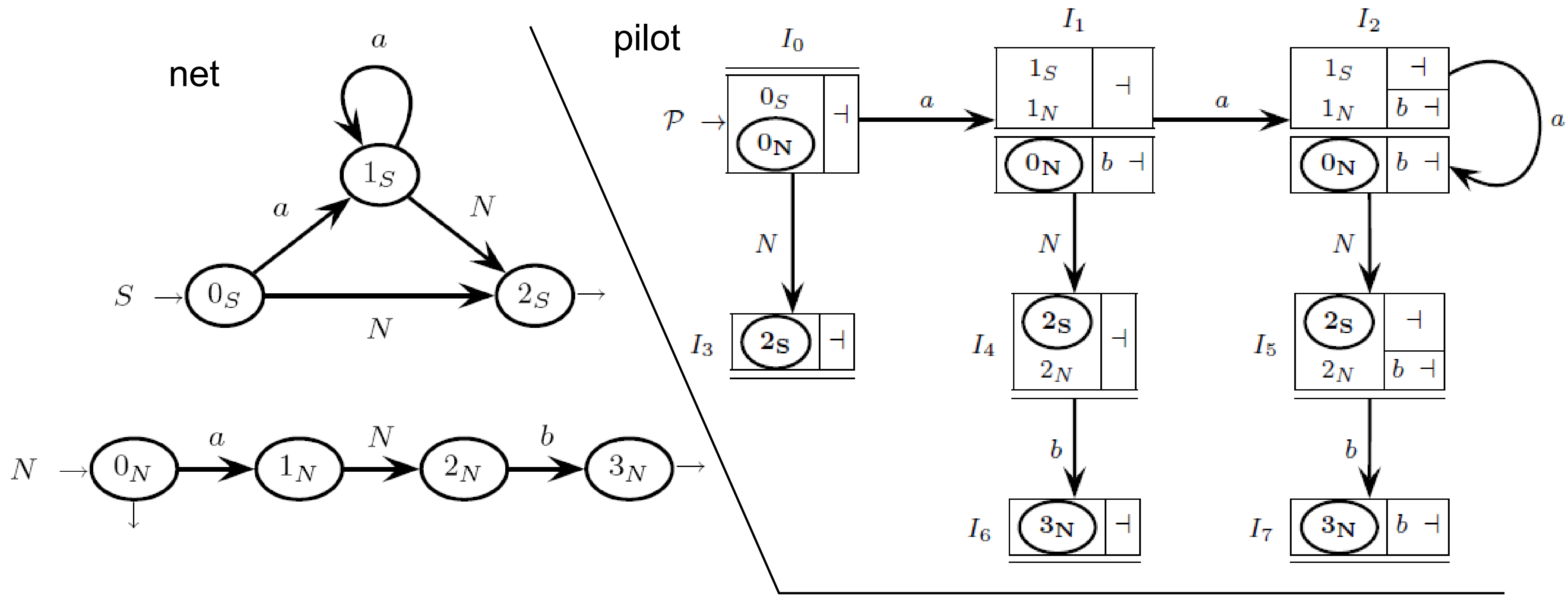
The analysis time complexity is $O(n) \leq k \times n + c$ for some integer constants k and c

The analysis space complexity is the max stack size, which is $n_T + n_N + 1 \leq k \times n + c$
(space complexity is always upper-bounded by time complexity)

A GRAMMAR WITH TWO ITEMS IN THE M-STATE BASES

It generates the deterministic language $\{ a^n b^m \mid n \geq m \geq 0 \}$

$$\begin{cases} S \to a^* \, N \\ N \to a \, N \, b \mid \varepsilon \end{cases}$$

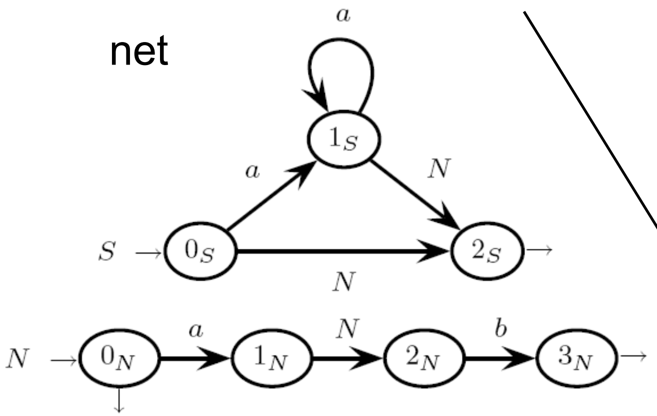


Two or more items in the m-state bases ⇒ the *PDA* carries on two or more parallel analysis threads, though there is only one final reduction because the condition *ELR* (1) is satisfied

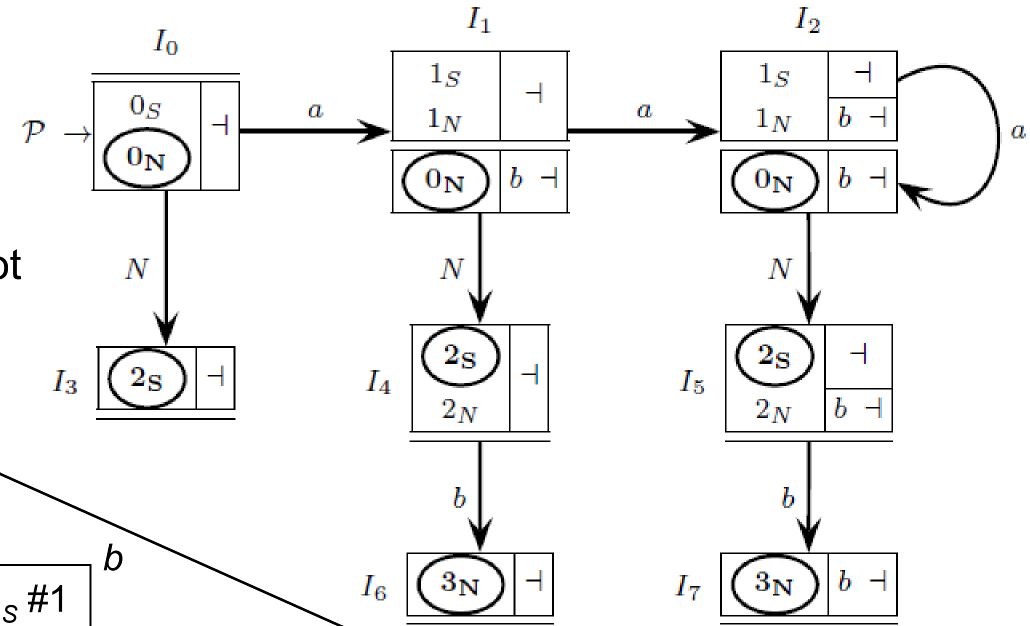
Try to simulate the analysis of a few sample strings: ε, *a a*, *a a b*, *a b*, *a b b*

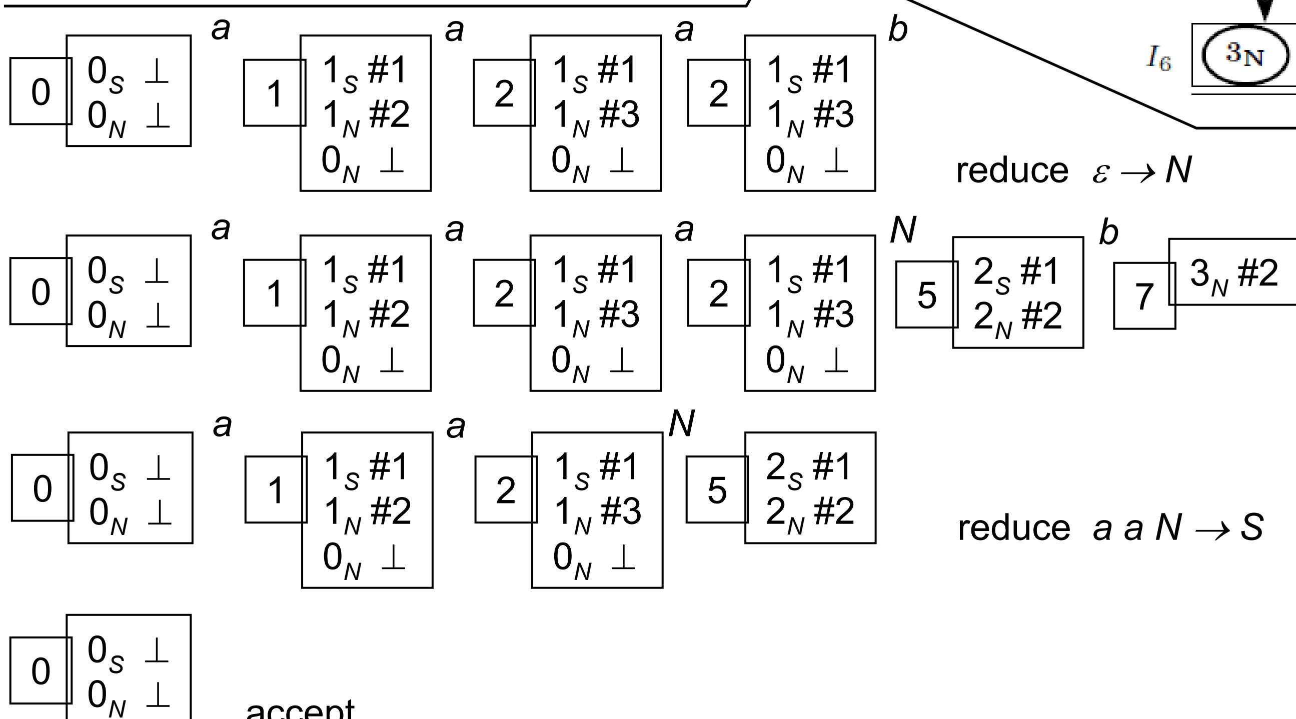
EXAMPLE – analysis of "*a a a b* ⊣ " uses all the net arcs

net



pilot





reduce $\varepsilon \rightarrow N$

reduce $a\ N\ b \rightarrow N$

PDA stack simulations

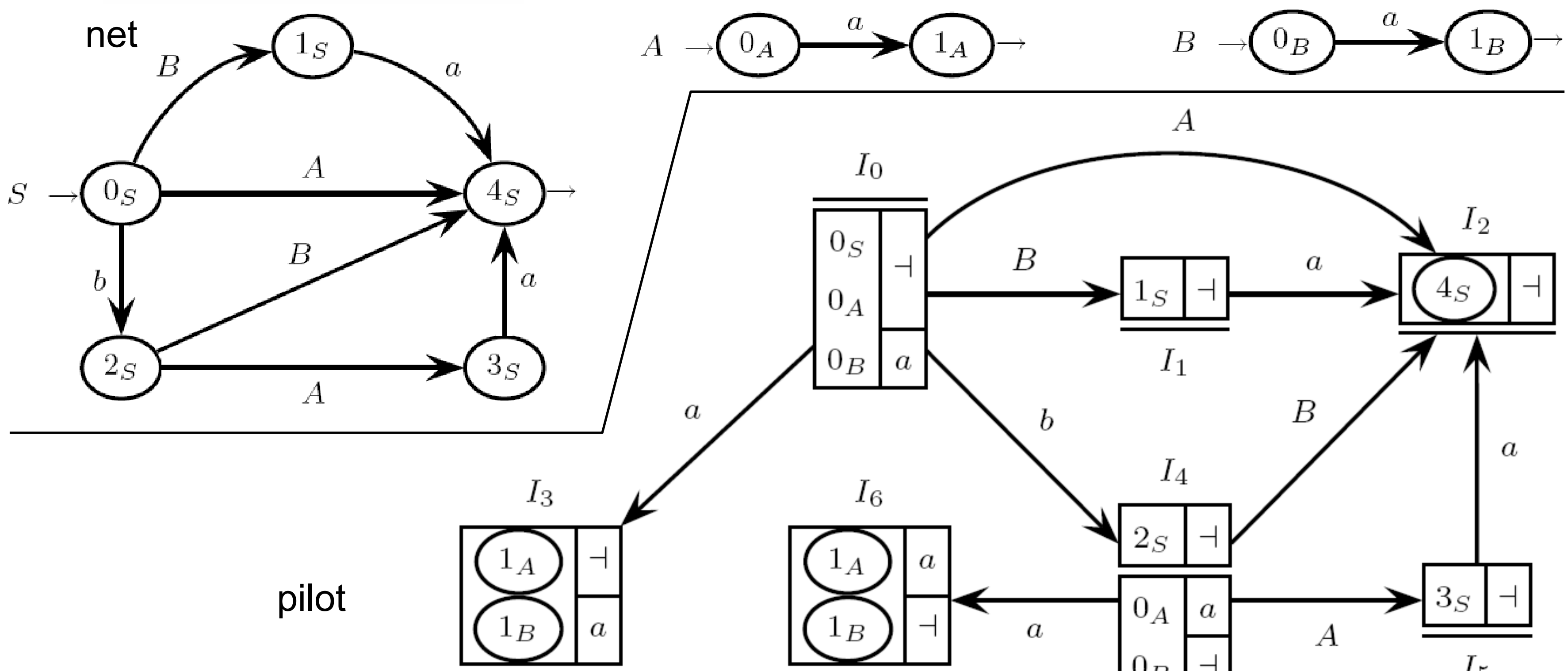
reduce $a\ a\ N \rightarrow S$

accept

A GRAMMAR THAT HAS TWO REDUCTIONS IN A M-STATE
yet that does not have any reduce-reduce conflicts

Finite language (grammar not recursive) $L = \{\ a,\ a\ a,\ b\ a\ a,\ b\ a\ \}$

$S \rightarrow A \mid Ba \mid bAa \mid bB$ $\quad A \rightarrow a$ $\quad B \rightarrow a$



The m-states that potentially violate the *ELR* (1) condition are I_3 and I_6

Yet none of them has outgoing arcs, so there are not any shift-reduce conflicts

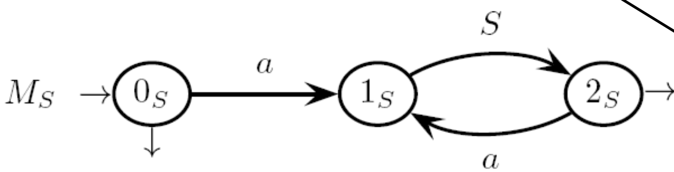
In I_3 and I_6 there are two reduction items with disjoint look-ahead $\Rightarrow$ no reduce-reduce conflict

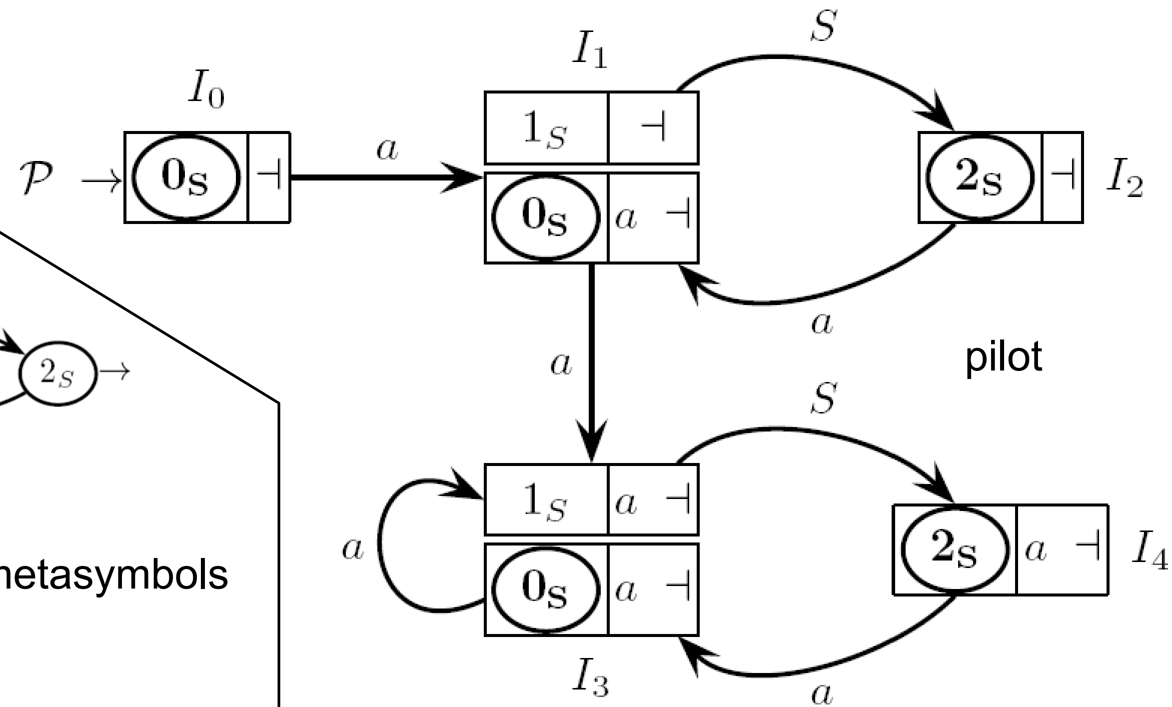
A GRAMMAR THAT VIOLATES THE CONDITION *ELR* (1) AS IT IS AMBIGUOUS

highly ambiguous rule

$$S \rightarrow (a\,S)^*$$

net





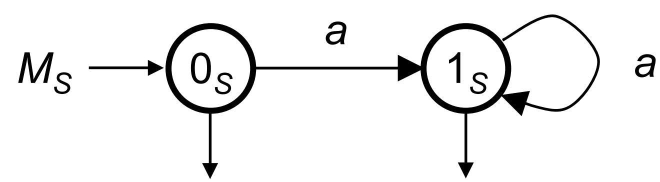
$L = a^*$ brackets "(", ")" are metasymbols

Various violations of determinism

- there are shift-reduce conflicts in the m-states I_1, I_3 and I_4
- char a is in the look-ahead of the reduction items 0_S and 2_S, as well as on an outgoing arc

A POSSIBLE REMEDY

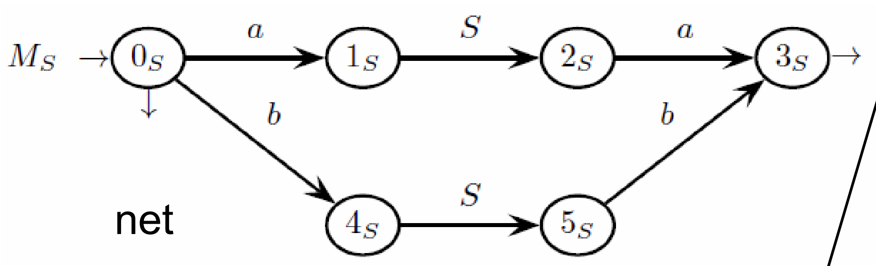
The language is $L = a^*$ and is regular. It is generated by grammar $S \rightarrow a^*$, which has a net like the one on the right that satisfies the condition *ELR* (1) (check it)

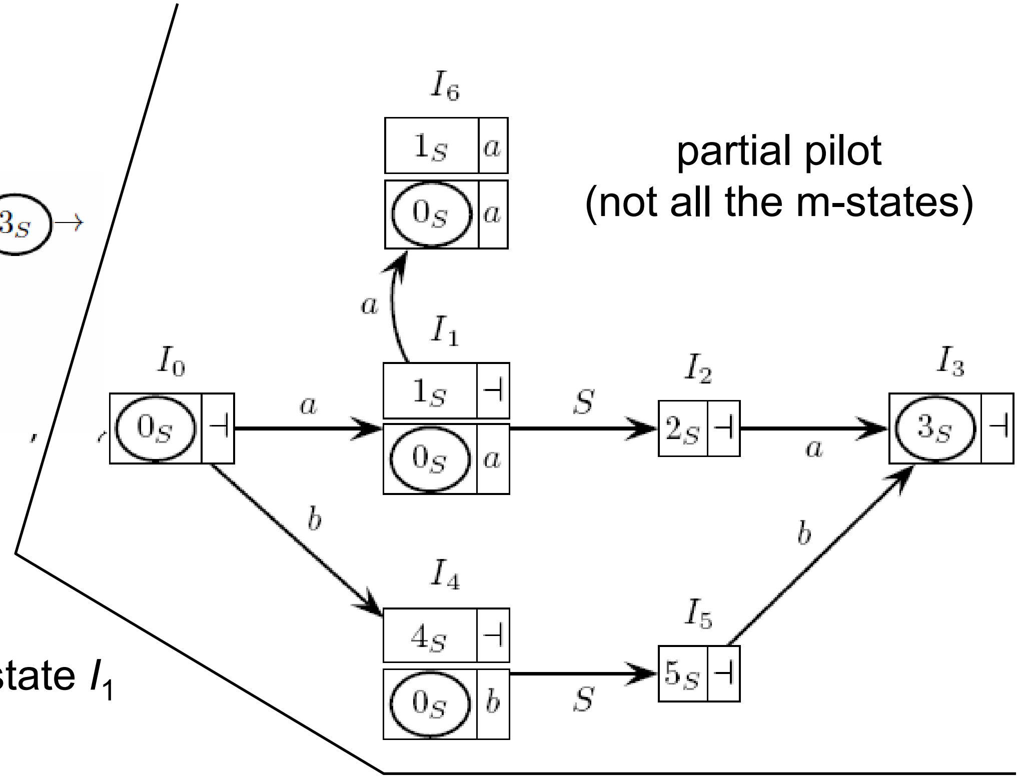


A GRAMMAR THAT VIOLATES THE CONDITION *ELR* (1) AS IT IS NON-DETERMINISTIC

palindrome

$$S \to aSa \mid bSb \mid \varepsilon$$





The language strings do not have a centre deterministically identifiable

There is a shift-reduce conflict in the m-state I_1 (and in I_4 as well though here not drawn)

Intuitively, after shifting a char *a* the *PDA* should (but is unable to) decide whether to

- *reduce*, if it has reached the palindrome centre
- *shift*, for the opposite reason (not in the centre)

Yet the *PDA* cannot know which case holds: the language is intrinsically non-deterministic

Such a language does not have any *ELR* (1) grammar, no matter how many rules are used

HOW TO IMPLEMENT THE *PDA* BY A VECTORED STACK

In an implementation of the *PDA* with some programming language, e.g., C, we can always mine the stack elements underneath the top one and directly look deep inside the stack

The third field of a stack item can be an integer that directly points back to the position of the stack element where the analysis thread begins

Actually the analyzer is no longer a true *PDA* as the stack alphabet becomes infinite

Such a variation is possible in every analyzer of practical interest and is not costly

In a closure item, write the current stack element index instead of the initial value “$\perp$”

In a base item, copy the same index value as that in the item before

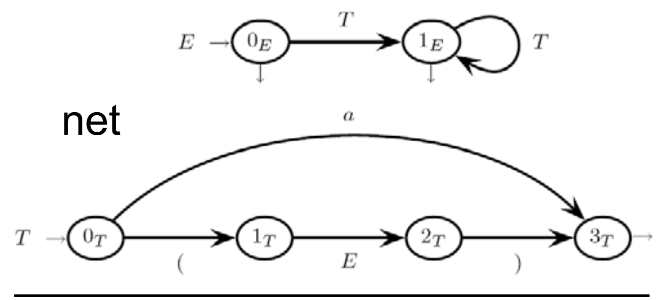
So the *PDA* does not scan back the reduction handle and goes directly to the origin

IT IS SIMILAR TO THE EARLEY ALGORITHM (see later …)

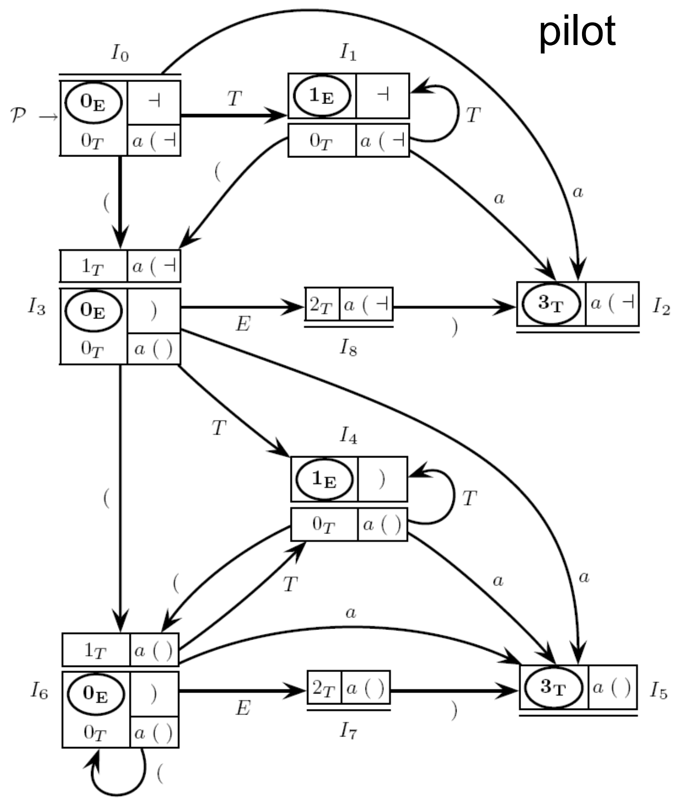
RUNNING EXAMPLE – vectored stack *PDA*

Analysis trace of string “ (( ) a) ”

net



pilot



stack base 0	1	2	3	4	5	⊣	effect after
$\boxed{0}$ 0_E 0, 0_T 0	(	(	)	a	)	⊣	initialisation of the stack
	(	(	)	a	)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1						shift on (
	(	(	)	a	)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{6}$ 1_T 13, 0_E 2, 0_T 2					shift on (
	(	(	E	) a	)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{6}$ 1_T 1, 0_E 2, 0_T 2	$\boxed{7}$ 2_T 1				reduction $\varepsilon \leadsto E$ and shift on E
	(	(	E	)	a	) ⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{6}$ 1_T 1, 0_E 2, 0_T 2	$\boxed{7}$ 2_T 1	$\boxed{5}$ 3_T 1			shift on)
	(	T		a	)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{4}$ 1_E 1, 0_T 2					reduction $(E) \leadsto T$ and shift on T
	(	T		a	)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{4}$ 1_E 1, 0_T 2		$\boxed{5}$ 3_T 2			shift on a
	(	T		T	)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{4}$ 1_E 1, 0_T 2		$\boxed{4}$ 1_E 1, 0_T 4			reduction $a \leadsto T$ and shift on T
	(	E			)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{8}$ 2_T 0					reduction $T\,T \leadsto E$ and shift on E
	(	E			)	⊣	
	$\boxed{3}$ 1_T 0, 0_E 1, 0_T 1	$\boxed{8}$ 2_T 0			$\boxed{2}$ 3_T 0		shift on)
	T					⊣	
	$\boxed{1}$ 1_E 0, 0_T 1						reduction $(E) \leadsto T$ and shift on T
	E					⊣	
							reduction $T \leadsto E$ and accept without shifting on E