Top-down Syntax Analysis
ELL (*k*) Method

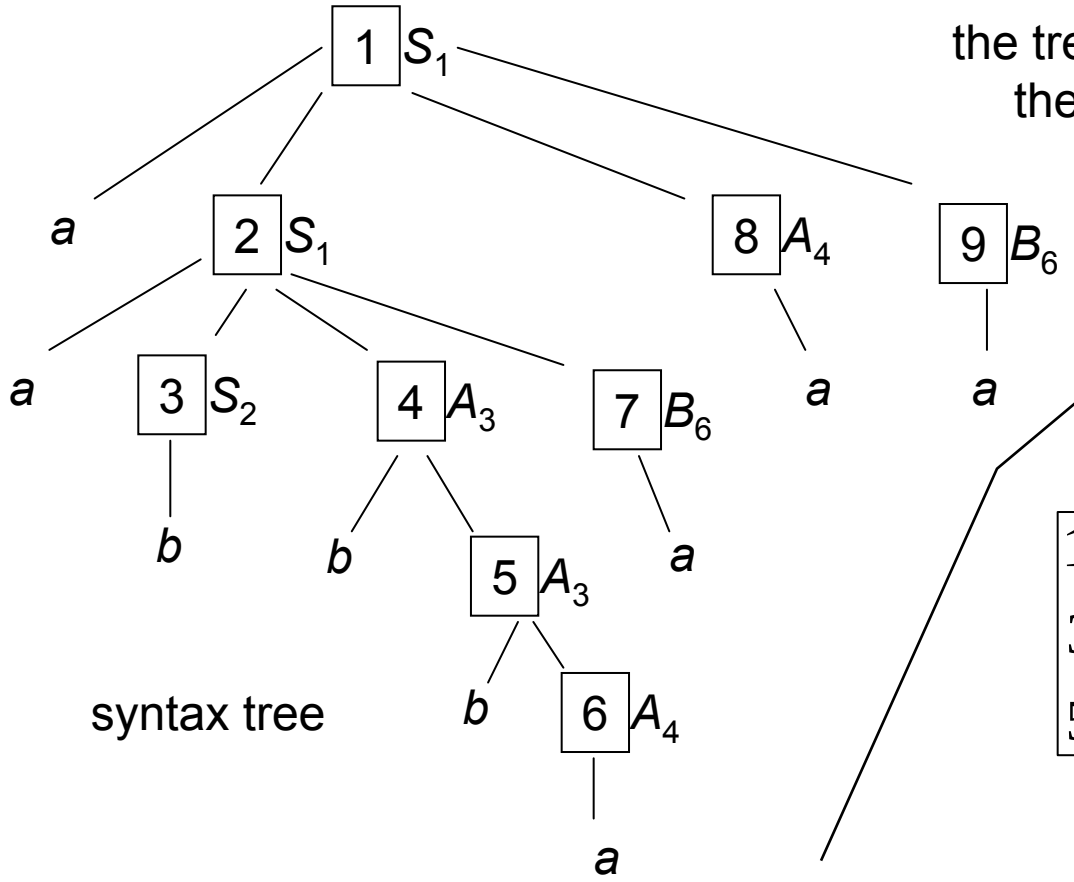
Translated and adapted by L. Breveglieri

TOP-DOWN ANALYSIS – 1

EXAMPLE – analysis of the valid phrase

a a b b b a a a a

The framed numbers indicate the application order of the grammar rules, and the non-terminal subscripts indicate the number of the applied rule



syntax tree

the tree is built top-down left-to-right
the derivation order is leftmost

BNF grammar

1. $S \to aSAB$	2. $S \to b$
3. $A \to bA$	4. $A \to a$
5. $B \to cB$	6. $B \to a$

TOP-DOWN ANALYSIS – 2

The analysis procedure is driven by the machine network – no need of a pilot graph, i.e., the machine net itself can directly work as a pilot for the syntax analyzer *PDA*

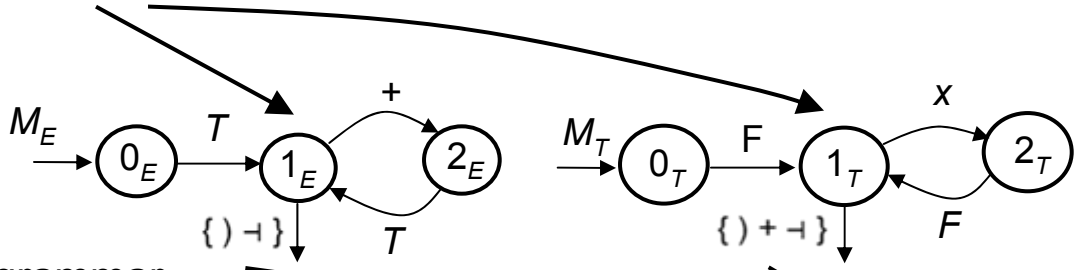
The syntax analyzer can recognize very early which rule it should apply to a phrase, in practice as soon as it finds the first (leftmost) character generated by a rule

We obtain the top-down analysis from the bottom-up one by some restrictions

Final target: the machine net itself will work as the "pilot graph" of the syntax analyzer

Yet the net has to be annotated, to choose which branch to take in the doubtful cases

in the bifurcation states, determinism wants disjoint guide sets



EBNF grammar

$E \rightarrow T\ (\ \text{“}+\text{”}\ T\ )^*$
$T \rightarrow F\ (\ \text{“}x\text{”}\ F\ )^*$
$F \rightarrow a \mid \text{“(“}\ E\ \text{“)”}$

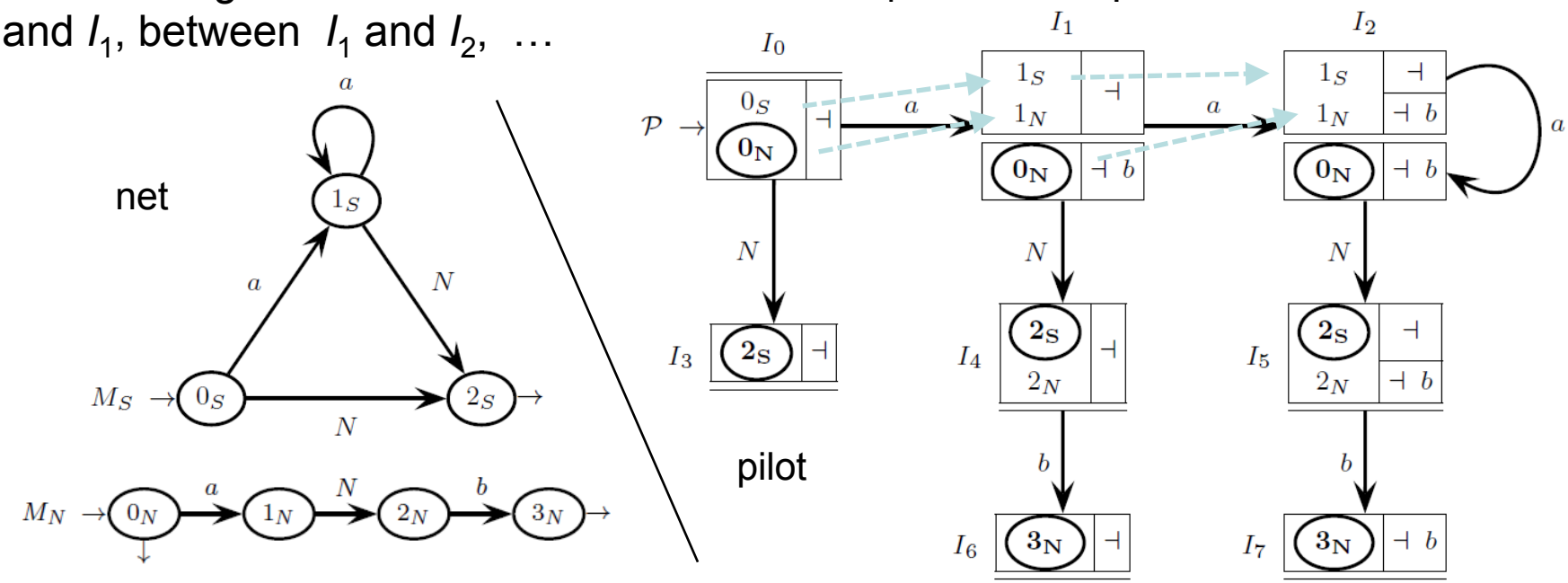
each arc (or dart) has a *guide set* that specifies which characters are expected when taking the arc

```
procedure T
  call F
  while (cc == '×')
    cc := next
    call F
  end
  if (cc ∈ { ) + -| })
    return
  else
    error
  endif
end T
```

RECALLS OF BOTTOM-UP ANALYSIS

A m-state *I* has a *multiple transition* if there are two (or more) items $\langle p, \pi\rangle$, $\langle r, \rho\rangle \in I$ and their two (or more) next states $\delta(p,X)$, $\delta(r,X)$ are both (or all) defined

EXAMPLE – grammar $S \rightarrow a^* N$ and $N \rightarrow a\ N\ b \mid \varepsilon$ has multiple transitions between I_0 and I_1, between I_1 and I_2, …



A multiple transition originates multiple analysis threads, which are typical in the *ELR* (1) analysis. They carry on in parallel different analysis hypotheses, therefore they are incompatible with top-down analysis

No multiple transitions: *Single Transition Property* (*STP*)

STP ensures there are not any convergent arcs in the pilot

LEFTMOST RECURSIVE DERIVATIONS

A derivation $A \Rightarrow^{+} A\,v$ with $v \neq \varepsilon$ is said to be *leftmost recursive*

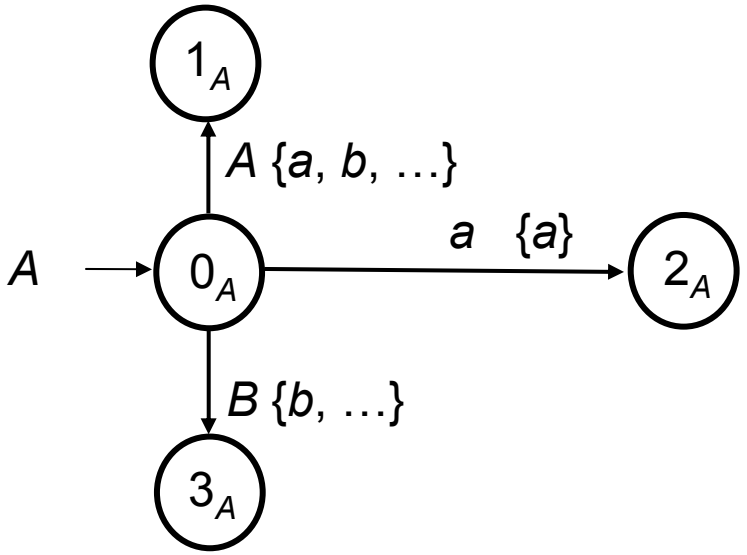
It is caused by the rules with a leftmost recursion, direct or indirect, possibly in the presence of *nullable nonterminals*

EXAMPLE $\quad E \to X\,E + a \mid a \qquad X \to \varepsilon \mid b$

Leftmost recursive derivations make top-down analysis impossible

EXAMPLE

$$A \to A.... \mid a.... \mid B$$
$$B \to b....$$



ELL (1) CONDITION FOR AN *EBNF* GRAMMAR

A grammar *G* (in general *EBNF*) represented as a machine net is *ELL* (1) if

1) it does not have any *leftmost recursive derivations*

2) its pilot graph satisfies the *ELR* (1) *condition*

3) its pilot graph does not have any *multiple transitions* (i.e., has the *STP*)

These three requirements are not completely independent of one another

In particular requirement (1) could be further restricted, but for simplicity it is better to keep all the three of them in the above form

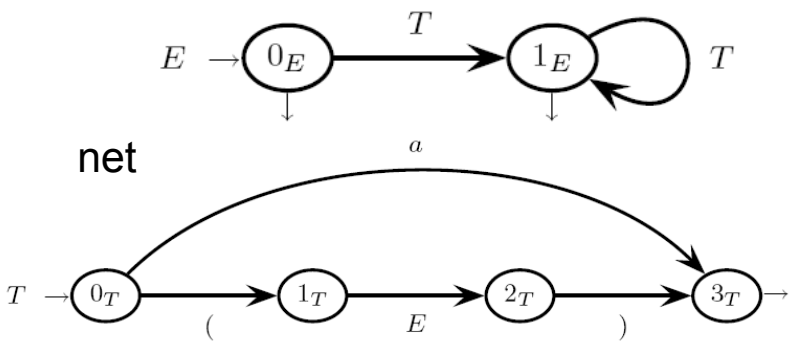
The *ELL* (1) condition above implies that the family of the *ELL* (1) grammars is contained in that of the *ELR* (1) grammars

The containment is strict; furthermore, also the family of *ELL* (1) languages is strictly contained in that of the *ELR* (1) (i.e., deterministic) languages

THE GRAMMAR OF THE RUNNING EXAMPLE IS *ELL* (1)

$E \to T^*$ *EBNF* grammar

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

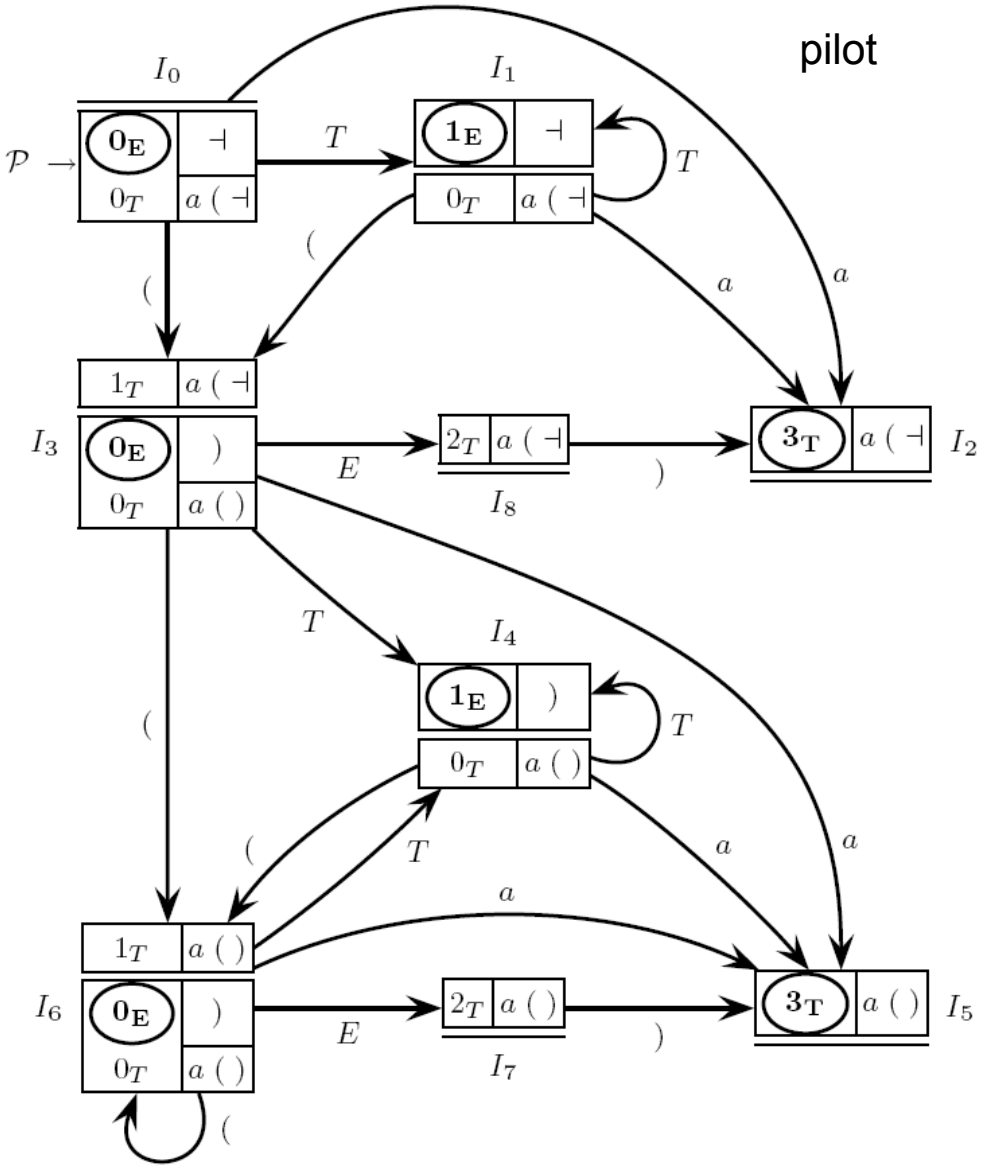


ELL (1) condition

no left recursion

pilot is *ELR* (1)

pilot has the *STP*



ELL (1) ANALYSIS TURNS OUT TO BE SUBSTANTIALLY SIMPLIFIED

ANTICIPATED (PREDICTIVE) DECISION – only one item in each m-state base

$\Rightarrow$ we know immediately which rule to apply (no need to wait for the reduction)

STACK POINTERS UNNECESSARY – once the rule to be applied is known

$\Rightarrow$ no need to carry on more than one analysis thread

$\Rightarrow$ unnecessary to keep the items corresponding to other analysis hypotheses

CONTRACTION OF THE STACK – only one analysis thread

$\Rightarrow$ no need to push on stack the state path followed

$\Rightarrow$ it suffices to push on stack the sequence of machines followed

FURTHER RESTRCTURING OF THE PILOT

$\Rightarrow$ separate the closure states (the initial states of the machines)

$\Rightarrow$ the pilot graph is isomorphic to the machine net

$\Rightarrow$ add some new arcs (*call arcs*) to the shift arcs

Call arc from q_A to 0_B if and only if machine M_A has a transition $q_A \rightarrow^B r_A$

Two or more initial states in the closure of a m-state originate a *call chain*

SIMPLIFICATION OF A PILOT THAT SATISFIES THE *ELL* (1) CONDITION

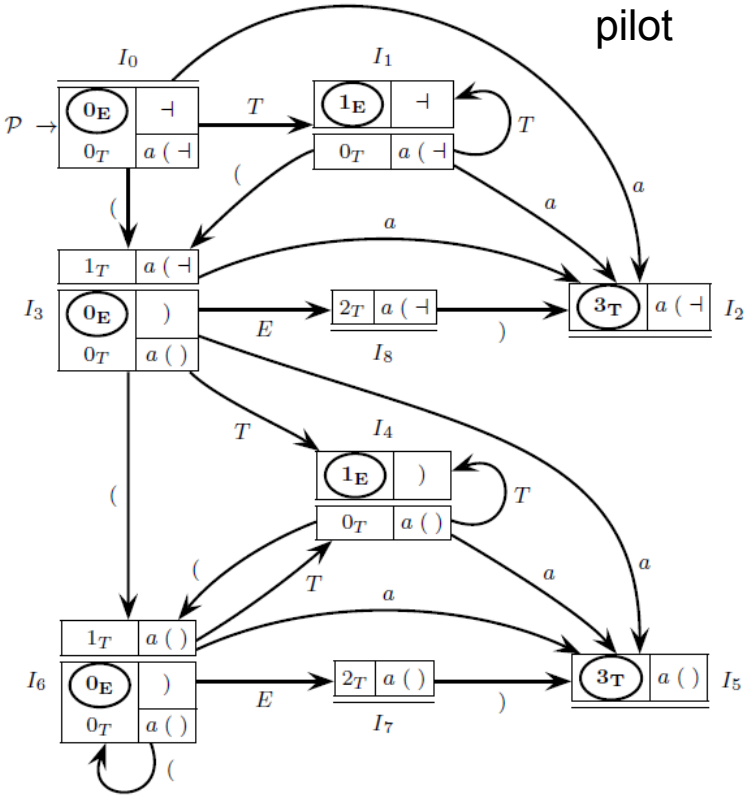
M-states with same kernel (kernel-identical) are unified (unify look-ahead) ⇒ pilot more compact

The kernel-identity relation is an equivalence relation

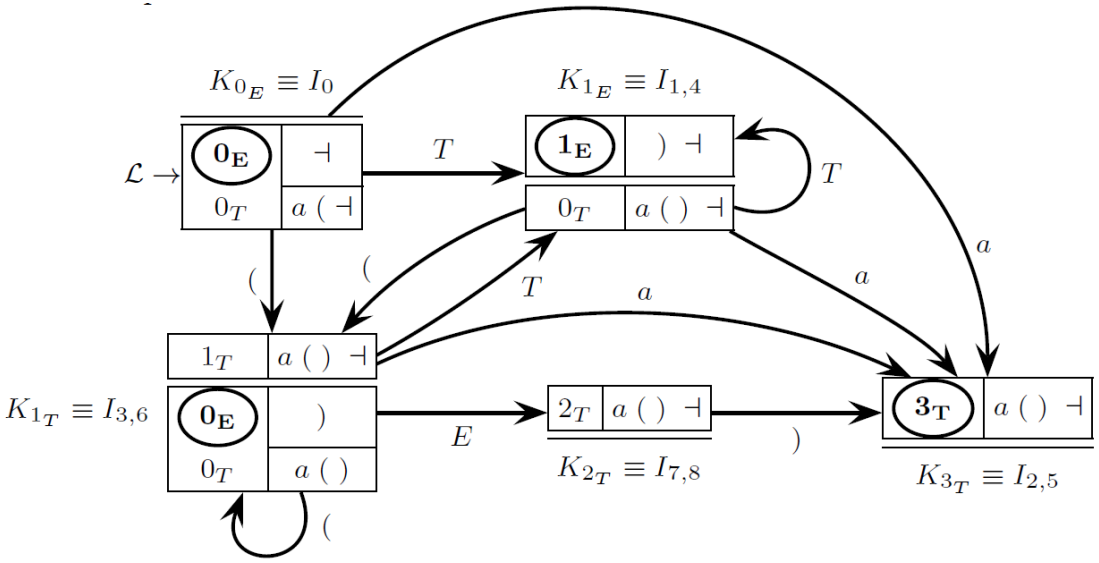
Transitions with same label from kernel-identical m-states go into kernel-identical m-states

The m-state bases (with non-initial states) contain only one item (consequence of STP)
⇒ they are in a one-to-one correspondence with the non-initial states of the machine net

pilot



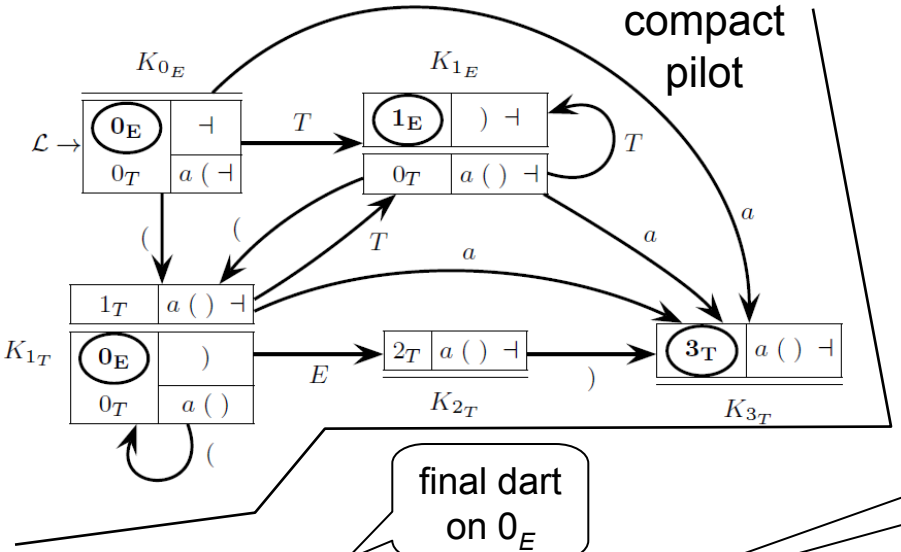
compact pilot



NOTE: the pilot graph “starts looking like” the machine net …

RESULTING PILOT AUTOMATON – PARSER CONTROL FLOW GRAPH (*PCFG*)

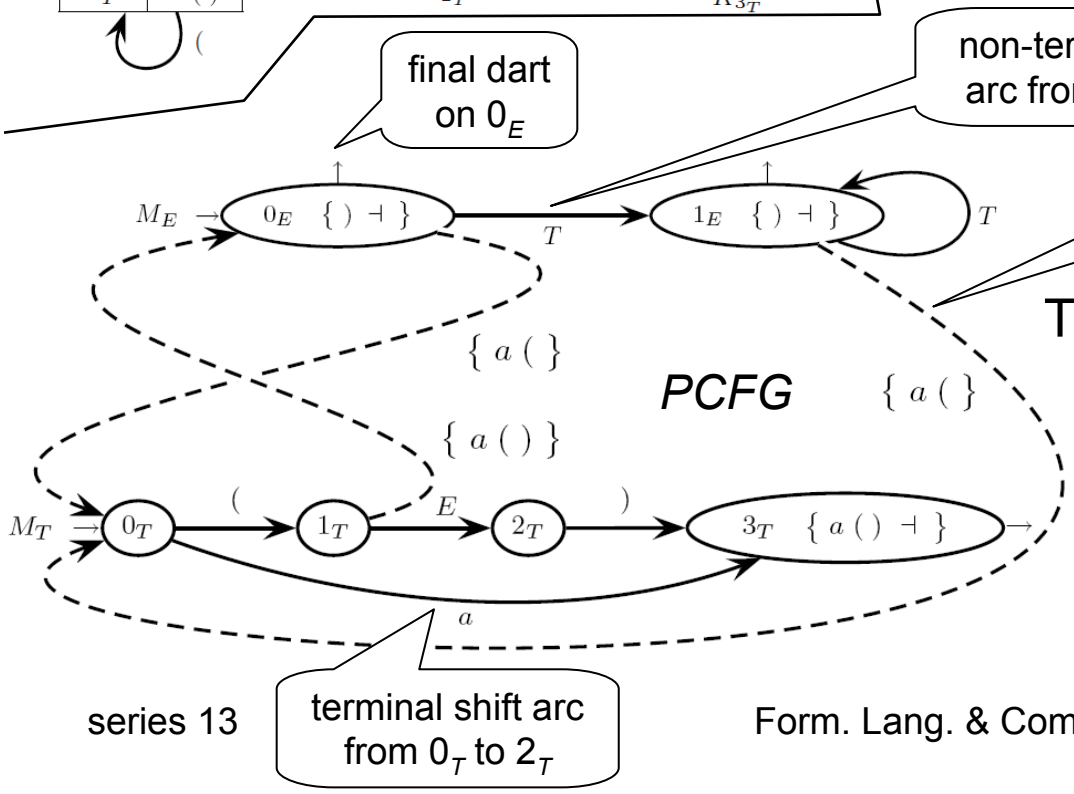
The Parser Control-Flow Graph (*PCFG*) is the control unit of the *ELL* (1) syntax analyzer *PDA*



the *prospect sets* are included only in the final states to choose whether to exit the machine or to continue with some more moves

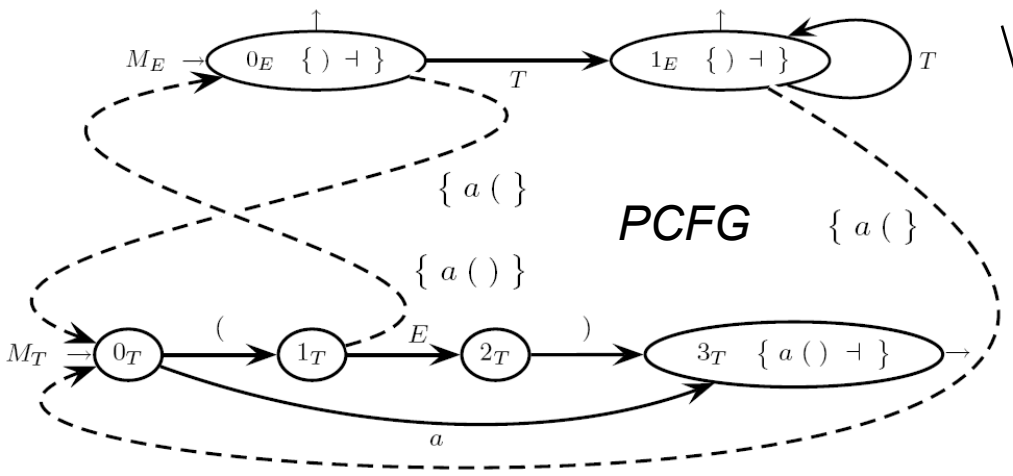
the call arcs are dashed and labeled with a *guide set*

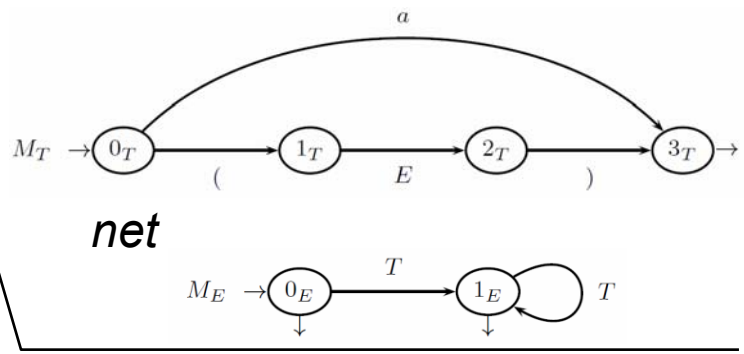
guide set = the characters that are expected in the input soon after calling the machine



The guide set allows to choose whether

- executing one call move
- scanning a terminal symbol
- executing one of two or more call moves
- exiting the machine (if final state)





how to define on a call arc $Gui\left(q_A \dashrightarrow 0_{A_1}\right)$

GUIDE SET

$$b \in Gui\left(q_A \dashrightarrow 0_{A_1}\right)$$

$$b \in Ini\left(L\left(0_{A_1}\right)\right)$$

$$A_1 \text{ is nullable and } b \in Ini\left(L\left(r_A\right)\right)$$

$$A_1 \text{ and } L\left(r_A\right) \text{ are both nullable and } b \in \pi_{r_A}$$

$$\exists \text{ in } \mathcal{F} \text{ a call arc } 0_{A_1} \overset{\gamma_2}{\dashrightarrow} 0_{A_2} \text{ and } b \in \gamma_2$$

The guide sets of the call arcs that depart from the same state have to

- be disjoint from one another, and
- be disjoint from all the scan arcs from the same state

Guide set on a dart of a final state f_A that contains an item $\langle f_A, \pi \rangle$

$Gui\ (f_A \rightarrow) = \pi$ the look-ahead π indicates the chars expected when M_A finishes

Trivially $Gui\ (p \rightarrow^a q) = a$ for a terminal shift arc, where a is a terminal symbol

SUMMARY

In a *PCFG* almost all the arcs (except the non-terminal shift) are interpreted as conditional instructions

Terminal arcs $p \rightarrow^{a} q$ run if an only if the current character $cc = a$

Call arcs $q_A \rightarrow 0_B$ run if an only if the current character is $cc \in Gui\ (q_A \rightarrow 0_B)$

Exit arcs (darts) $f_A \rightarrow$ from a state with an item $\langle f_A, \pi \rangle$ run if and only if the current character is $cc \in \pi$

The non-terminal arcs $p \rightarrow^{A} q$ are interpreted as (unconditioned) return instructions from a machine

It is possible to prove that

disjoint guide sets (in the bifurcation states) $\Leftrightarrow$ condition *ELL* (1) satisfied

PREDICTIVE TOP-DOWN ANALYZER – ALGORITHM

The stack elements are the states of the *PCFG*, i.e., the net states

The stack is initialized with element $\langle 0_E \rangle$

Suppose $\langle q_A \rangle$ is the stack top – it means that machine M_A is active and in the state q_A

The *ELL* syntax analyzer (*PDA*) has four move types

scan move if the shift arc $q_A \xrightarrow{cc} r_A$ exists, then scan the next token and replace the stack top by $\langle r_A \rangle$ (the active machine does not change)

call move if there exists a call arc $q_A \overset{\gamma}{\dashrightarrow} 0_B$ such that $cc \in \gamma$, let $q_A \xrightarrow{B} r_A$ be the corresponding nonterminal shift arc; then pop, push element $\langle r_A \rangle$ and push element $\langle 0_B \rangle$

return move if q_A is a final state and token cc is in the prospect set associated with q_A, then pop

recognition move if M_A is the axiom machine, q_A is a final state and $cc = \dashv$, then accept and halt

In any other case the analyzer stops and rejects the input string

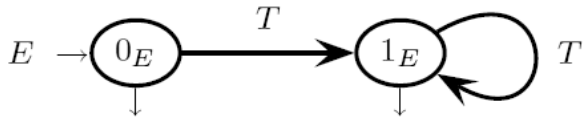
string x = (a)

stack	x	*predicate*	
$\langle 0_E \rangle$	(a) ⊣	$(\in \gamma = \{ a (\}$	
$\langle 1_E \rangle \langle 0_T \rangle$	(a) ⊣	scan	
$\langle 1_E \rangle \langle 1_T \rangle$	a) ⊣	$a \in \gamma = \{ a (\}$	← two-step call move
$\langle 1_E \rangle \langle 2_T \rangle \langle 0_E \rangle$	a) ⊣	$a \in \gamma = \{ a (\}$	← two-step call move
$\langle 1_E \rangle \langle 2_T \rangle \langle 1_E \rangle \langle 0_T \rangle$	a) ⊣	scan	
$\langle 1_E \rangle \langle 2_T \rangle \langle 1_E \rangle \langle 3_T \rangle$	) ⊣	$) \in \pi = \{ a ( ) \dashv \}$	
$\langle 1_E \rangle \langle 2_T \rangle \langle 1_E \rangle$	) ⊣	$) \in \pi = \{) \dashv \}$	
$\langle 1_E \rangle \langle 2_T \rangle$	) ⊣	scan	
$\langle 1_E \rangle \langle 3_T \rangle$	⊣	$\dashv \in \pi = \{ a ( ) \dashv \}$	
$\langle 1_E \rangle$	⊣	$\dashv \in \pi = \{) \dashv \}$	accept

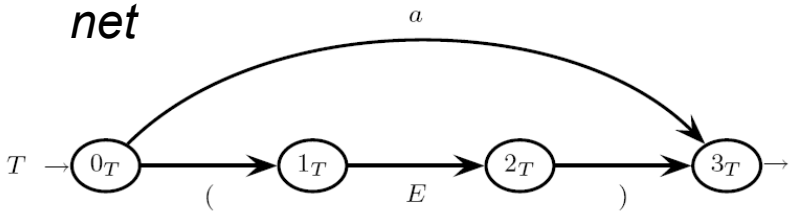
analysis trace of string x = ( )

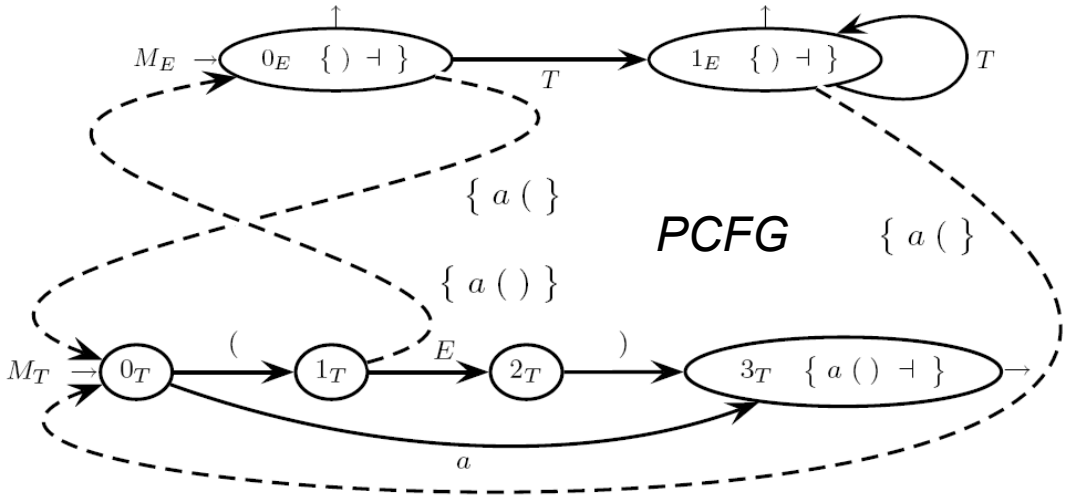
0_E	()⊣	call
$1_E\ 0_T$	()⊣	
$1_E\ 1_T$	)⊣	call
$1_E\ 2_T\ 0_E$	)⊣	return
$1_E\ 2_T$	)⊣	
$1_E\ 3_T$	⊣	return
1_E	⊣	accept

ANALYSIS EXAMPLES



net





analysis trace of string x = a

0_E	a ⊣	call
$1_E\ 0_T$	a ⊣	
$1_E\ 3_T$	⊣	return
1_E	⊣	accept

LEFTMOST DERIVATION – RIGHT LINEARIZED GRAMMAR

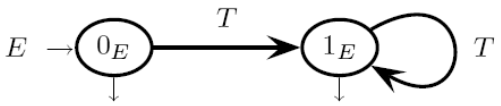
stack	x	*leftmost derivation of the right linearized grammar*
$\langle 0_E \rangle$	$(\, a\,)\, \dashv$	$0_E \Rightarrow 0_T\, 1_E$
$\langle 1_E \rangle\, \langle 0_T \rangle$	$(\, a\,)\, \dashv$	$0_E \stackrel{+}{\Rightarrow} (\, 1_T\, 1_E$
$\langle 1_E \rangle\, \langle 1_T \rangle$	$a\,)\, \dashv$	$0_E \stackrel{+}{\Rightarrow} (\, 0_E\, 2_T\, 1_E$
$\langle 1_E \rangle\, \langle 2_T \rangle\, \langle 0_E \rangle$	$a\,)\, \dashv$	$0_E \stackrel{+}{\Rightarrow} (\, 0_T\, 1_E\, 2_T\, 1_E$
$\langle 1_E \rangle\, \langle 2_T \rangle\, \langle 1_E \rangle\, \langle 0_T \rangle$	$a\,)\, \dashv$	$0_E \stackrel{+}{\Rightarrow} (\, a\, 3_T\, 1_E\, 2_T\, 1_E$
$\langle 1_E \rangle\, \langle 2_T \rangle\, \langle 1_E \rangle\, \langle 3_T \rangle$	$)\, \dashv$	$0_E \stackrel{+}{\Rightarrow} (\, a\, \varepsilon\, 1_E\, 2_T\, 1_E$
$\langle 1_E \rangle\, \langle 2_T \rangle\, \langle 1_E \rangle$	$)\, \dashv$	$0_E \stackrel{+}{\Rightarrow} (\, a\, \varepsilon\, 2_T\, 1_E$
$\langle 1_E \rangle\, \langle 2_T \rangle$	$)\, \dashv$	$0_E \stackrel{+}{\Rightarrow} (\, a\,)\, 3_T\, 1_E$
$\langle 1_E \rangle\, \langle 3_T \rangle$	$\dashv$	$0_E \stackrel{+}{\Rightarrow} (\, a\,)\, \varepsilon\, 1_E$
$\langle 1_E \rangle$	$\dashv$	$0_E \stackrel{+}{\Rightarrow} (\, a\,)\, \varepsilon$

$$E \Rightarrow T \Rightarrow (\, E\,) \Rightarrow (\, T\,) \Rightarrow (\, a\,)$$

leftmost derivation of the EBNF grammar

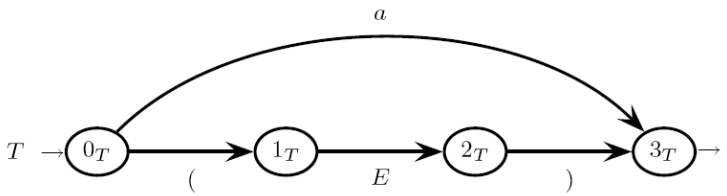
$$E \to T^*$$
$$T \to \text{`('}\ E\ \text{`)'} \mid a$$

EBNF grammar



$$0_E \to 0_T\ 1_E \mid \varepsilon$$
$$1_E \to 0_T\ 1_E \mid \varepsilon$$

right linear grammar of M_E



$$0_T \to a\ 3_T \mid \text{`('}\ 1_T$$
$$2_T \to \text{`)'}\ 3_T$$
$$1_T \to 0_E\ 2_T$$
$$3_T \to \varepsilon$$

right linear grammar of M_T

IMPLEMENTNG THE PARSER BY MEANS OF RECURSIVE PROCEDURES

Each machine becomes a procedure without parameters
$\Rightarrow$ we have one procedure per each non-terminal

Call move $\Rightarrow$ procedure call

Scan move $\Rightarrow$ call procedure *next* (the interface to the lexical analyzer or scanner)

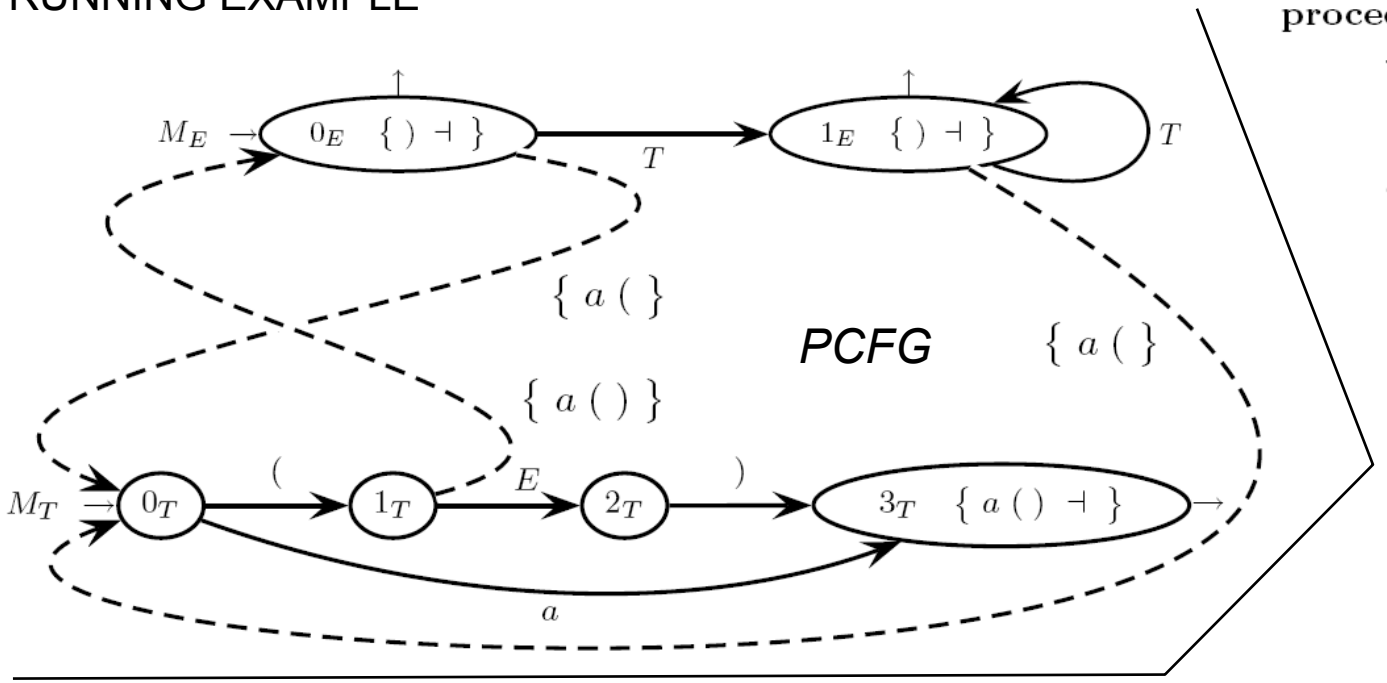
Return move $\Rightarrow$ return from procedure

Parser Control Flow Graph becomes the control graph of the procedure

Use the guide sets and the prospect sets to choose which move to execute

The analysis starts by calling the axiomatic procedure

Technique also known as recursive descent



RECURSIVE DESCENT PARSER

```
program ELL_PARSER
        cc = next
        call E
        if cc ∈ { ⊣ } then accept
        else reject end if
end program
```

```
procedure E
        - - optimized
        while cc ∈ { a ( } do
                call T
        end while
        if cc ∈ { ) ⊣ } then
                return
        else
                error
        end if
end procedure
```

```
procedure T
        - - state 0_T
        if cc ∈ { a } then
                cc = next
        else if cc ∈ { ( } then
                cc = next
                - - state 1_T
                if cc ∈ { a ( ) } then
                        call E
                else
                        error
                end if
                - - state 2_T
                if cc ∈ { ) } then
                        cc = next
                else
                        error
                end if
        else
                error
        end if
        - - state 3_T
        if cc ∈ { a ( ) ⊣ } then
                return
        else
                error
        end if
end procedure
```

DIRECT SIMPLIFIED CONSTRUCTION OF THE *PCFG*

No need to build the full *ELR* pilot graph and derive the *PCFG* from it

Directly put call arcs on the machine net and annotate the net with the *prospect* and *guide* sets

Sets of recursive equations to compute the prospect sets and then the guide sets

prospect set π

- we (re)use the rules already seen for computing the look-ahead sets of *ELR*
- the recursive rules below compute a prospect set for each state of the *PCFG*, though the ELL condition uses only those in the final states (with a final dart)

For an initial state 0_A

$$\pi_{0_A} := \pi_{0_A} \cup \bigcup_{q_i \xrightarrow{A} r_i} \Big(Ini\big(L(r_i)\big) \cup \textbf{if } Nullable\big(L(r_i)\big) \textbf{ then } \pi_{q_i} \textbf{ else } \emptyset \Big)$$

the same chars in π may originate from different terms of the equation

For any other state *q*

$$\pi_q := \bigcup_{p_i \xrightarrow{X_i} q} \pi_{p_i}$$

initialization – set to ┤ the prospect set of the initial state of the axiomatic machine and to empty all the other prospect sets

GUIDE SET *Gui* – essentially the same clauses of the already seen definition

Equation for the guide set of a call arc (uses the prospect set)

$$Gui(q_A \dashrightarrow 0_{A_1}) := \bigcup \left\{ \begin{array}{l} Ini\big(L(A_1)\big) \\ \hline \textbf{if } Nullable(A_1) \textbf{ then } Ini\big(L(r_A)\big) \\ \textbf{else } \emptyset \textbf{ endif} \\ \hline \textbf{if } Nullable(A_1) \wedge Nullable\big(L(r_A)\big) \textbf{ then } \pi_{r_A} \\ \textbf{else } \emptyset \textbf{ endif} \\ \hline \bigcup_{0_{A_1} \dashrightarrow 0_{B_i}} Gui(0_{A_1} \dashrightarrow 0_{B_i}) \end{array} \right.$$

the same chars in *Gui* may come from different terms of the equation

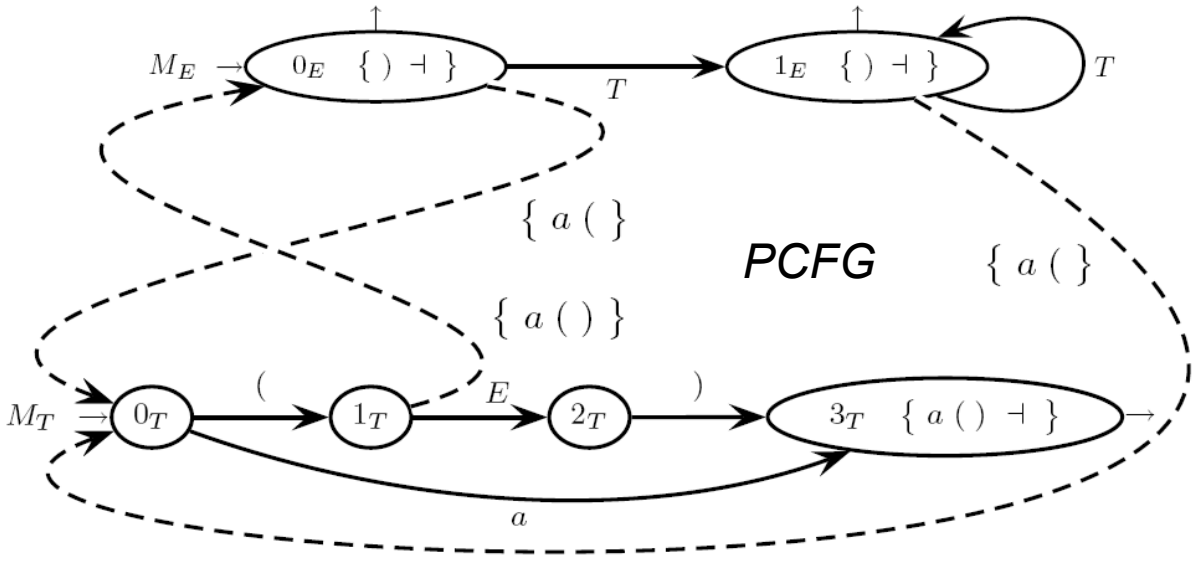
Furthermore (for the final darts and the terminal shift arcs)

$$Gui(f_A \rightarrow) := \pi_{f_A} \qquad Gui\left(q_A \xrightarrow{a} r_A\right) := \{\, a \,\}$$

f_a is a final state

initialization – set to empty all the guide sets

RUNNING EXAMPLE – VERY FEW ITERATIONS SUFFICE



1. first iteratively compute the prospect sets
2. then iteratively compute the guide set

prospect sets

$1_T \xrightarrow{E} 2_T \Rightarrow \pi_{0_E} := \pi_{0_E} \cup Ini(2_T) = \{\dashv\} \cup \{)\} = \{\dashv)\}$

$0_E \xrightarrow{T} 1_E \;\; 1_E \xrightarrow{T} 1_E \Rightarrow \pi_{0_T} := \pi_{0_T} \cup Ini(1_E) \cup \pi_{0_E} = \emptyset \cup \{(a\} \cup \{\dashv)\} = \{(a \dashv)\}$

guide sets

$Gui(0_E \rightarrow 0_T) := Ini\,(T) = \{\,(\,a\,\}$ $\qquad$ $Gui(1_E \rightarrow 0_T) := Ini\,(T) = \{\,(\,a\,\}$

$Gui(1_T \rightarrow 0_E) := Ini\,(E) \cup Ini(2_T) \cup Gui(0_E \rightarrow 0_T) = \{(a\} \cup \{)\} \cup \{(a\} = \{(a)\}$

Prospect sets of						*Guide sets of*		
0_E	1_E	0_T	1_T	2_T	3_T	0_E --→ 0_T	1_E --→ 0_T	1_T --→ 0_E
⊣	∅	∅	∅	∅	∅	∅	∅	∅
⊣)	⊣)	(a ⊣)	(a ⊣)	(a ⊣)	(a ⊣)	(a	(a	(a)
⊣)	⊣)	(a ⊣)	(a ⊣)	(a ⊣)	(a ⊣)	(a	(a	(a)

LENGTHENING THE PROSPECTION FOR GRAMMARS THAT ARE NOT *ELL* (1)

If the *ELL* (1) condition is not verified, we can try to modify the grammar and make it of type *ELL* (1)

This approach may take a long time and be a hard work

- sometimes easy, e.g., turning left into right recursion
- or when finding out that the language is regular …

Alternative approach – use a longer look-ahead

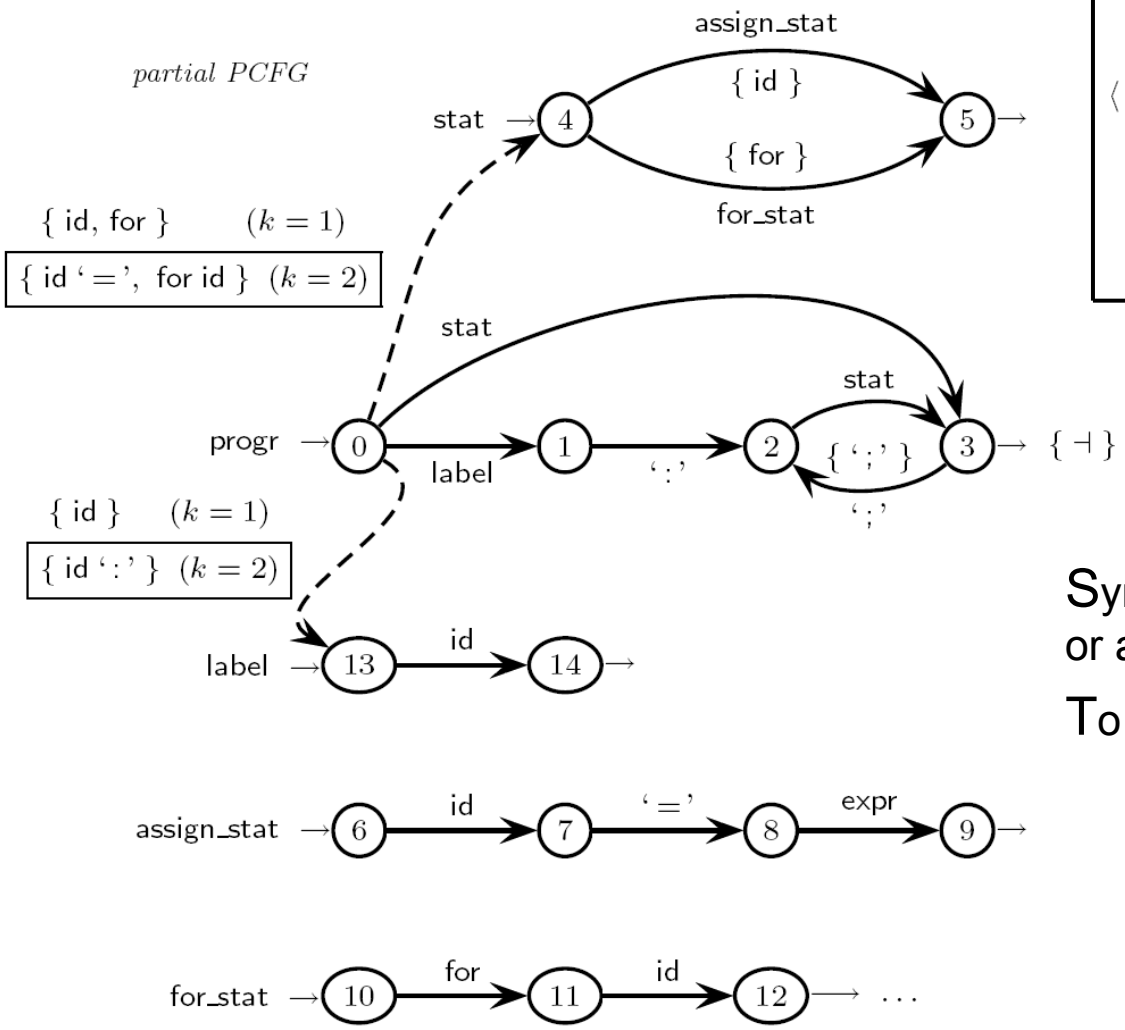
the analyzer looks at a number $k > 1$ of consecutive characters (or tokens) in the input

if the guide sets of length k on alternative moves are disjoint, then the *ELL* (k) analysis is possible

Example with look-ahead length $k = 2$

EXAMPLE – conflict between instruction labels and variable names

$$\langle \text{progr} \rangle \rightarrow [\ \langle \text{label} \rangle\ \text{':'}\ ]\ \langle \text{stat} \rangle\ (\ \text{';'}\ \langle \text{stat} \rangle\ )^*$$
$$\langle \text{stat} \rangle \rightarrow \langle \text{assign\_stat} \rangle\ |\ \langle \text{for\_stat} \rangle\ |\ \ldots$$
$$\langle \text{assign\_stat} \rangle \rightarrow \text{id}\ \text{'='}\ \langle \text{expr} \rangle$$
$$\langle \text{for\_stat} \rangle \rightarrow \text{for id}\ \ldots$$
$$\langle \text{label} \rangle \rightarrow \text{id}$$
$$\langle \text{expr} \rangle \rightarrow \ldots$$



NOTE – the guide sets of length $k = 2$ are framed

Symbol "id" may be an instruction label or a variable to be assigned

To distinguish, look at the next token

if "id :" $\Rightarrow$ label

if "id =" $\Rightarrow$ assignment