Recognition of Free Languages

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

RECOGNITION OF FREE LANGUAGES

RECOGNIZERS: from abstract automa to parsers
(to construct the syntax tree of the phrase).

In the regular languages structure can be predetermined (right or left linear).

In the free languages, instead, structure can have different forms.

AUTOMATON MODEL TO RECOGNISE FREE LANGUAGES:

- has finite states and a stack memory
- the moves are in correspondence with the rules of the grammar
- it is not always possible to have a deterministic automaton
- the presence of both finite states and a memory stack makes everything more complex to analyze

In the following:

- pushdown automaton model
- pushdown deterministic automaton (for det. free languages)
- fast parsing algorithms (that work in linear time, e.g. LL(k) and LR(k))
- and a general recognition algorithm (e.g. Earley)

PUSHDOWN AUTOMATON MODEL

stack top

$$A_1 A_2 \ldots A_k$$

current char

$$a_1 a_2 \ldots \quad a_i \quad \ldots a_n \dashv$$

1) has an auxiliary memory, structured as a unlimited stack of symbols
2) input string (or source string)
3) applicable operations:
   push(B), *push*(B_1, B_2, ... B_n): stacks the symbols on the top (on the right of A_k)
   emptiness check: the predicate *empty* is true if and only if $k = 0$
   pop: removes A_k from the stack top, unless the stack is empty
4) Z_0 is the *stack bottom* symbol (can only be read)
5) -| is the *end marker* symbol of the input string
6) *configuration*: current state, current char, stack contents

MOVE OF THE AUTOMATON:

- read the current input character and shift the input head of one position to the right, or execute a spontaneous move without reading anything in the input string and without shifting the input head
- read the stack top symbol and cancel it, or read Z_0 if the stack is empty
- depending on the current character and the stack top symbol, determine the new state to go to and possibly push a sequence of stack symbols

HOW TO TRANSFORM A GRAMMAR INTO A PUSHDOWN AUTOMATON

1) Grammar rules can be viewed as the instructions of a non-deterministic pushdown automaton (with only one finite state). Intuitively: such an automaton works in a *goal oriented* way and uses the stack as a notebook of the sequence of actions to undertake in the next future.
2) The stack symbols can be both terminal and non-terminal symbols of the grammar. If the stack contains the sequence A_1, ... A_k, the automaton executes first the action associated with A_k, which should recognise whether in the input string, starting from the position of the currrent character a_i, there is a string *w* that can be derived from A_k; if it is so, the action shifts the input head of $|w|$ positions.
3) An action can be recursively divided into a series of sub-actions, if to recognise the non-terminal symbol A_k it is necessary to recognise other terminal or non-terminal symbols.

The initial action is the grammar axiom: the pushdown recogniser must in fact check whether the source string can be derived from the axiom. Initially the stack contains only the symbol Z_0 and the axiom S, and the input head is positioned in correspondence with the initial (leftmost) character of the input string. At every step the automaton chooses (non-deterministically) one applicable rule and executes the corresponding move. The input string is recognised (accepted) if and only if when it has been scanned completely, the stack is empty.

GIVEN A GRAMMAR $G = (V, \Sigma, P, S)$

Rule	Move	Comment

$A \to BA_1 ... A_n \quad n \geq 0$	**if** *top* = *A* **then** *pop*;
	  push(A_n ... A_1B)
	end if
		to recognise A one must recognise B A_1 ... A_n

$A \to bA_1 ... A_n \quad n \geq 0$	**if** *cur-char* = *b* ∧ *top* = *A*
	  then *pop*; *push*(A_n ... A_1);
	  shift input head
	end if
		b is the first expected char and is read; A_1 ... A_n are still to be recognised

$A \to \varepsilon$	**if** *top* = *A* **then** *pop* **end if**	ε is recognised, and it derives from A

for every char $b \in \Sigma$	**if** *cur-char* = *b* ∧ *top* = *b*
	  then *pop*; *shift input head*
	end if
		b is the first expected char and is read

— — —	**if** *cur-char* = -| ∧ *stack empty*
	  then *accept string*
	end if halt
		the input string is scanned, and the stack does not contain any more actions

EXAMPLE – rules and moves of the goal oriented language recognizer

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

Rule	Move
1. $S \rightarrow aS$	**if** *cur-char* = *a* ∧ *top* = S **then** *pop*; *push*(S); *shift* **end if**
2. $S \rightarrow A$	**if** *top* = S **then** *pop*; *push*(A) **end if**
3. $A \rightarrow aAb$	**if** *cur-char* = *a* ∧ *top* = A **then** *pop*; *push*(*b*A); *shift* **end if**
4. $A \rightarrow ab$	**if** *cur-char* = *a* ∧ *top* = A **then** *pop*; *push(b*); *shift* **end if**
5.	**if** *cur-char* = *b* ∧ *top* = *b* **then** *pop*; *shift* **end if**
6.	**if** *cur-char* = -| ∧ *empty stack* **then** *shift* **end if** *halt*

Choosing move 1 or 2 is non-deterministic, as move 2 could be selected also when the current input character is *a*, and the same holds for 3 and 4.

The automaton recognises a string if and only if it is generated by the grammar: for every successful automaton computation there exists a corresponding grammar derivation, and viceversa. Therefore the automaton simulates the *leftmost derivations* of the grammar.

$S \Rightarrow A \Rightarrow aAb \Rightarrow aabb$

successful computation

stack	x
Z_0S	$aabb$-|
$Z_0\mathrm{A}$	$aabb$-|
Z_0bA	abb-|
Z_0bb	bb-|
Z_0b	b-|
Z_0	-|

However, the automaton can not guess which the correct derivation will be; it need examine all possibilities.

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaA \Rightarrow errore$

$S \Rightarrow aS \Rightarrow aA \Rightarrow aaAb \Rightarrow errore$

$S \Rightarrow aS \Rightarrow aA \Rightarrow aab \Rightarrow errore$

$S \Rightarrow A \Rightarrow ab \Rightarrow errore$

A string is accepted by two or more different computations if and only if the grammar is ambiguous.

The previous construction (from grammar to automaton) can be inverted, and can be used to obtain the grammar starting from the pushdown automaton, if this is one-state. Therefore, if follows that:

PROPERTY – the family of free languages, generated by free grammars, coincides with the family of the languages recognised by one-state pushdown automata (in general non-deterministic).

NOTE: the construction does not work for multi-state pushdown automata. However the property holds in general, for automata with any number of states, although the proof needs more sophisticated concepts and tools (see the bibliography).

UNFORTUNATELY THE ABOVE CONSTRUCTED PUSHDOWN AUTOMATON IS NON-DETERMINISTIC (in general) AND EXPLORES ALL THE MOVES THAT ARE APPLICABLE AT ANY POINT, AND THEREFORE HAS EXPONENTIAL TIME COMPLEXITY WITH RESPECT TO THE LENGTH OF THE SOURCE STRING ... however, more efficient algorithms exist ... to see next

DEFINITION OF THE PUSHDOWN AUTOMATON

A pushdown automaton M (in general non-deterministic) is defined by:

1. Q	finite state set of the control unit
2. Σ	input alphabet
3. Γ	stack alphabet
4. δ	transition function
5. $q_0 \in Q$	initial state
6. $Z_0 \in \Gamma$	initial stack symbol (stack bottom symbol)
7. $F \subseteq Q$	set of final states (may be empty, too)

TRANSITION FUNCTION:

domain:	$Q \times (\Sigma \times \varepsilon) \times \Gamma$
image:	the finite subsets of $Q \times \Gamma^*$

READ MOVE: if the automaton is in the state *q* and Z is on the top of the stack, when *a* is read it can go to one of the states p_i $1 \le i \le n$, after executing the operations *pop* and *push*(γ_i).

$$\boxed{\begin{array}{l}\delta(q,a,Z)=\{(p_1,\gamma_1),(p_2,\gamma_2),\ldots(p_n,\gamma_n)\} \\ \text{with } n\ge 1, a\in\Sigma, Z\in\Gamma \quad \text{with } p_i\in Q,\gamma_i\in\Gamma^*\end{array}}$$

NOTE: the choice of the *i*-th action of the *n* possible ones, is non-deterministic; shift is automatic; the stack top symbol is always popped; the sequence of symbols to push may be empty.

SPONTANEOUS MOVE: if the automaton is in the state *q* and Z is on the top of the stack, without reading any input character it can go to one of the states p_i, $1 \le i \le n$, after executing the operations *pop* and *push*(γ_i).

$$\boxed{\begin{array}{l}\delta(q,\varepsilon,Z)=\{(p_1,\gamma_1),(p_2,\gamma_2),\ldots(p_n,\gamma_n)\} \\ \text{with } n\ge 1, a\in\Sigma, Z\in\Gamma \quad \text{with } p_i\in Q,\gamma_i\in\Gamma^*\end{array}}$$

NON-DETERMINISM: it depends on the image of the transition function, which is a set of possible options, and on the presence of spontaneous moves.

INSTANTANEOUS CONFIGURATION OF THE AUTOMATON M: is a triple

$$\boxed{(q, y, \eta) \in Q \times \Sigma^* \times \Gamma^+}$$

that describes:

- q the current state of the control unit
- *y* the suffix of the input string *x*, still to be read
- *η* the current sequence of symbols contained in the stack

INITIAL CONFIGURATION: (q_0, $\times$, Z_0)
A CONFIGURATION (*q*, ε, η) OR (*q*, -|, η) IS <u>FINAL</u> IF $q \in F$

TRANSITION BETWEEN TWO CONFIGURATIONS: $\boxed{(q, y, \eta) \to (p, z, \lambda)}$

CHAIN OF ONE OF MORE TRANSITIONS: $\boxed{\overset{+}{\to}}$

Previous conf.	Next conf.	Executed move
$(q, az, \eta Z)$	$(p, z, \eta\gamma)$	read move $\delta(q, a, Z) = \{(p, \gamma), \ldots\}$
$(q, az, \eta Z)$	$(p, az, \eta\gamma)$	spontaneous move $\delta(q, \varepsilon, Z) = \{(p, \gamma), \ldots\}$

Although a move always cancels the stack top symbol, this symbol can be pushed back soon after, thus actually leaving the stack top unchanged.

Machine M recognizes (accepts) a string *x* by final state if:

$$\boxed{(q_0, x, Z_0) \overset{+}{\rightarrow} (q, \varepsilon, \lambda)}$$

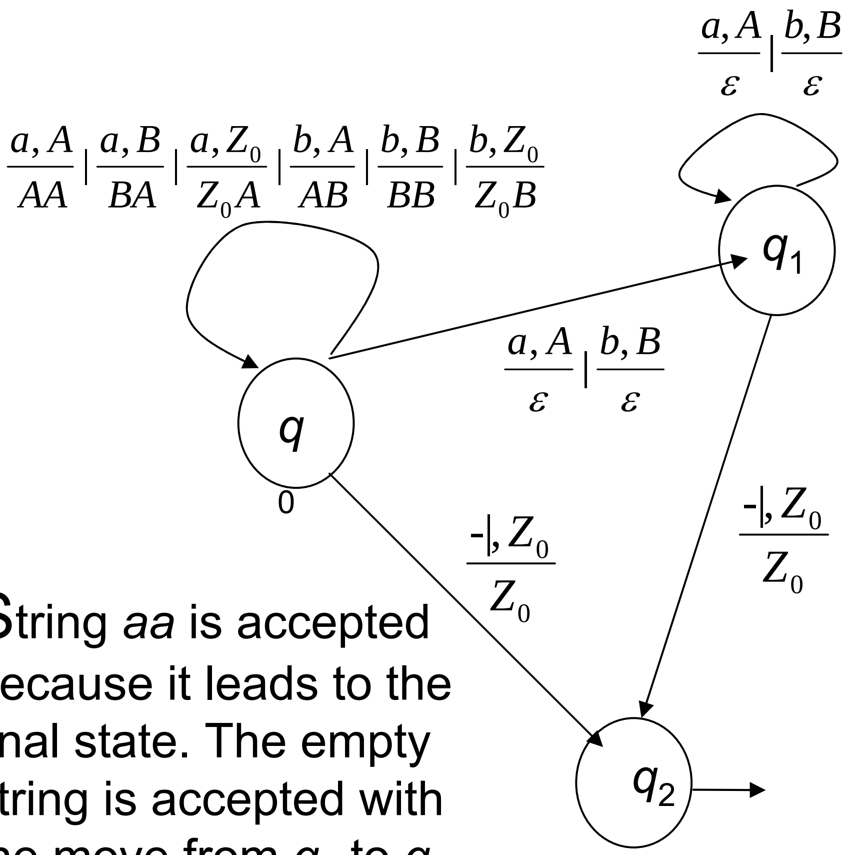
where *q* is a final state and $\lambda \in \Gamma^*$ is the final stack contents, which in general may be not empty.

STATE-TRANSITION GRAPH OF A PUSHDOWN AUTOMATON

EXAMPLE: palindromes of even length, accepted by the pushdown recognizer by final state.

$$\boxed{L = \left\{ uu^R \mid u \in \{a, b\}^* \right\}}$$

Stack	x	State	Comment
Z_0	*aa*-|	q_0	
Z_0A	*a*-|	q_0	
Z_0AA	-|	q_0	reject: no move is defined for (q_0, -|, A)



$\frac{a,A}{AA} \mid \frac{a,B}{BA} \mid \frac{a,Z_0}{Z_0A} \mid \frac{b,A}{AB} \mid \frac{b,B}{BB} \mid \frac{b,Z_0}{Z_0B}$
$\frac{a,A}{\varepsilon} \mid \frac{b,B}{\varepsilon}$
$\frac{a,A}{\varepsilon} \mid \frac{b,B}{\varepsilon}$
q_1
q
0
$\frac{-|,Z_0}{Z_0}$
$\frac{-|,Z_0}{Z_0}$
q_2

Stack	x	State	Comment
Z_0	*aa*-|	q_0	
Z_0A	*a*-|	q_0	
Z_0	-|	q_1	
Z_0	-|	q_2	recognition by final state

String *aa* is accepted because it leads to the final state. The empty string is accepted with the move from q_0 to q_2.

VARIETIES OF PUSHDOWN AUTOMATA

The pusdown automaton defined in the previous example differs from that directly obtainable from the grammar, for two reasons: it has two or more states and uses a different acceptance condition (final state instead of empty stack).

POSSIBLE ACCEPTANCE MODES:

- by final state (accepts when enters a final state, independently of the contents of the stack)

or

- by empty stack (accepts when the stack gets empty, independently of the current state)

or

- combined (by final state and empty stack)

PROPERTY – for the family of (non-deterministic) pushdown automata with states, the three acceptance modes listed above are equivalent, that is the three families of recognised languages coincide.

OPERATION WITHOUT SPONTANEOUS LOOPS AND ON-LINE

A generic pushdown automaton can execute an unlimited number of moves without reading any input character. This may happen if and only if the automaton enters a loop made only of spontaneous moves. Such a behaviour might prevent the automaton of reading completely the input string, or cause the automaton to execute an unlimited number of moves before deciding whether to accept or to reject the string. Both behaviours are clearly undesirable for a practical perspective.

It is always possible to construct an equivalent automaton without spontaneous loops.

A pushdown automaton operates in on-line mode if, as soon as it has read the last character of the input string, it can decide immediately whether to accept or to reject the string, without executing any other move. Clearly on-line mode is a desirable behaviour from a practical perspective.

It is always possible to construct an equivalent automaton operating in on-line mode

FREE LANGUAGES AND PUSDOWN AUTOMATA: ONE FAMILY

It can be proved that the language accepted by a generic pushdown automaton with states is free, and as it is already known that every free language can be recognised by a (non-deterministic one-state) pushdown automaton, it follows that:

$\boxed{\text{PROPERTY – the family LIB of free languages coincides with that of the languages recognised by pushdown automata (without any kind of restriction).}}$

And more specifically:

$\boxed{\text{PROPERTY – the family LIB of free languages coincides with that of the languages recognised by the pushdown automata with one state only, but in general non-deterministic.}}$

RECOGNISING THE INTERSECTION OF A FREE AND A REGULAR LANGUAGE

It is now easy to justify (if not to prove rigorously) that the intersection of a free and a regular language is free as well:

Given a grammar G and a finite state automaton A, the pushdown automaton M that recognizes the intersection L(G) ∩ L(A) can be obtained as follows:

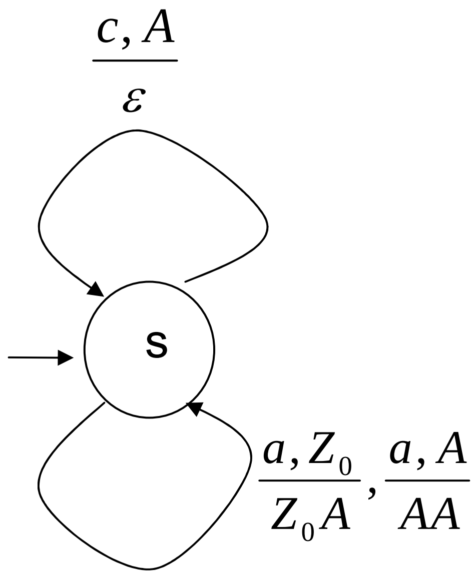
1) construct the one-state pushdown automaton N recognizing L(G) by empty stack
2) construct the pushdown automaton (with states) M, the state-transition graph of which is the cartesian product of those of N and A, applying the cartesian product construction so that the actions of M on the stack are the same as those of N

The obtained pushdown automaton M:

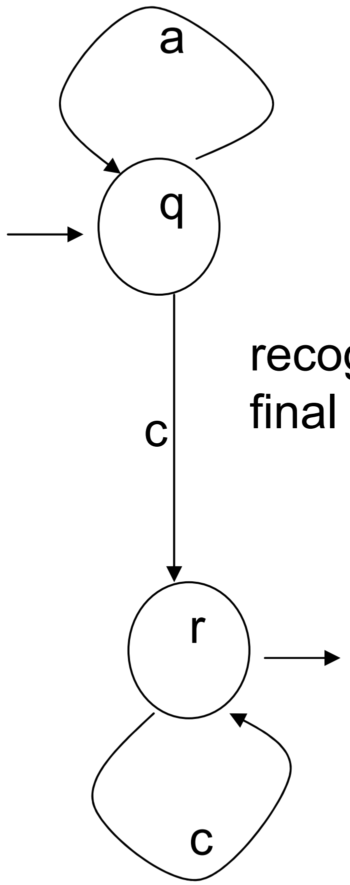
- as its states, has pairs of states of the component machines N and A
- accepts by final state and empty stack (combined acceptance mode)
- the states containing a final state of A are themselves final
- is deterministic, if both component machines N and A are so
- accepts by final state all and only the strings belonging to the intersection language

EXAMPLE

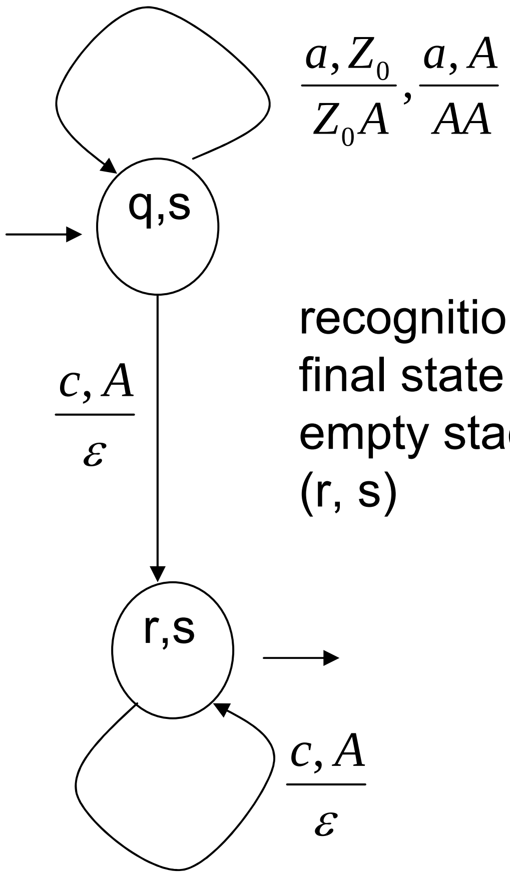
$$\boxed{L_{Dyck} \bigcap (a^* c^+) = \left\{ a^n c^n \mid n \geq 1 \right\}}$$


$\frac{c, A}{\varepsilon}$
s
$\frac{a, Z_0}{Z_0 A}, \frac{a, A}{AA}$

recognition by empty stack



recognition by final state


$\frac{a, Z_0}{Z_0 A}, \frac{a, A}{AA}$
q,s
$\frac{c, A}{\varepsilon}$
r,s
$\frac{c, A}{\varepsilon}$

recognition by final state and empty stack (r, s)

PUSHDOWN AUTOMATA AND DETERMINISTIC LANGUAGES

Deterministic languages are of special interest for compilation, therefore they deserve a more detailed study.

A pushdown automaton can exhibit three different forms of non-determinism:

1. uncertainty between two or more read moves: this happens if for some triple (q, a, A) of the domain of the transition function δ, there are two or more images: $| \delta(q, a, A) | > 1$
2. uncertainty between one read and one spontaneous move: this happens when both $\delta(q, a, A)$ and $\delta(q, \varepsilon, A)$ are defined
3. uncertainty between two or more spontaneous moves: this happens if for some pair (a, A) the transition function $\delta(q, \varepsilon, A)$ has two or more images: $| \delta(q, \varepsilon, A) | > 1$

If the transition function δ exhibits none of the three forms 1, 2, 3 of non-determinism listed above, the pushdown automaton is DETERMINISTIC and the recognised language is said to be deterministic as well. The family of the languages recognised by deterministic pushdown automata is called DET.

EXAMPLE

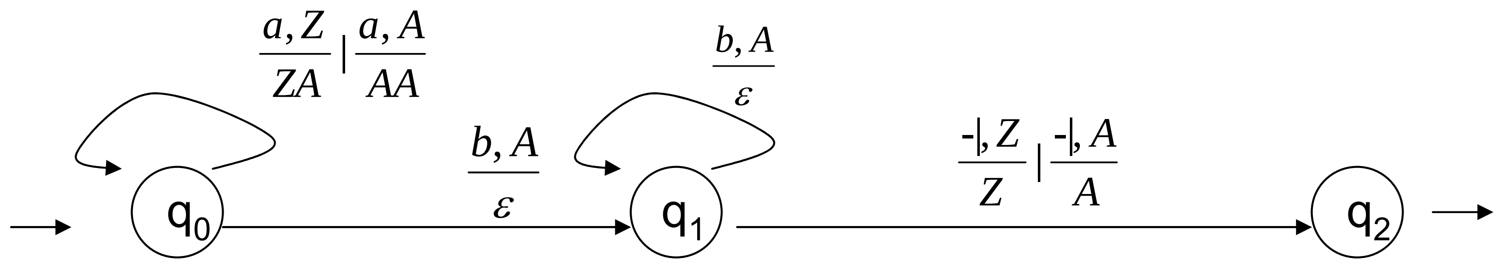
$L = \{a^n b^m \mid n \geq m > 0\}$

forms of non-determinism

1. $\delta(q_0, a, A) = \{(q_0, b), (q_0, bA)\}$
2. $\delta(q_0, \varepsilon, S) = \{(q_0, A)\} \quad \delta(q_0, a, S) = \{(q_0, S)\}$

The same language is recognised in det. way by the automaton:

$M_2 = (\{q_0, q_1, q_2\}, \{a, b\}, \{A, Z\}, \delta, q_0, Z, \{q_2\})$



$\frac{a,Z}{ZA} \mid \frac{a,A}{AA}$
$\frac{b,A}{\varepsilon}$
$\frac{b,A}{\varepsilon}$
$\frac{\dashv,Z}{Z} \mid \frac{\dashv,A}{A}$
q_0
q_1
q_2

Intuitively: M reads *a* and pushes it encoded as A; when M reads the first *b*, M pops one A and moves to q_1; then for each *b* that M reads, M pops one A; if there were more *b* than *a* characters, M would fail; when the end-marker is encountered, M moves to q_2 and accepts, independently of the stack top symbol.

CLOSURE PRPERTIES OF DETERMINISTIC LANGUAGES

Name L, D and R a language belonging to one of the families LIB, DET and REG, respectively.

Operation	Property	Already known property
Mirroring	$D^R \notin DET$	$D^R \in LIB$
Star	$D^* \notin DET$	$D^* \in LIB$
Complement	$\neg D \in DET$	$\neg L \notin LIB$
Union	$D_1 \bigcup D_2 \notin DET, D \bigcup R \in DET$	$D_1 \bigcup D_2 \in LIB$
Concatenation	$D_1.D_2 \notin DET, D.R \in DET$	$D_1.D_2 \in LIB$
Intersection	$D \cap R \in DET$	$D_1 \cap D_2 \notin LIB$

NON-DETERMINISTIC LANGUAGES

It is not always possible to recognise a free language by means of a deterministic pushdown automaton (differently to what happens with regular languages).

Observe that:

$1)\ DET \subseteq LIB$ (a det. automaton is a special case of a generic one)

$2)\ DET \neq LIB$ (some closure properties hold true for either family, not for both nor for none)

One comes to the conclusion that:

the family DET of deterministic free languages is strictly contained in that of free languages.

EXAMPLE: (non-deterministic) union of deterministic languages

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

L is a *inherently* non-det. language (no det. automaton could ever recognise L).

In fact, a det. automaton recognising L should push each character *a* that is read, encoding it in some way (e.g. *aabb*) and, if the string is in the former set, the aut. should pop one *a* after reading each *b*, or, if the string belongs to the latter set, the aut. should pop one *a* after reading each pair of characters *b*. However the det. automaton can not know ahead of time which the right choice will be, and waiting for reading all the characters *b* would be too late to be still of some utility. Therefore such a det. automaton may not exist or, said differently, it must undertake both ways and therefore it must behave in a non-deterministic way.

DETERMINISM AND INAMBIGUITY OF THE LANGUAGE

If a language is accepted by a deterministic pushdown automaton, every phrase is recognised by a single computation. On the other side, the grammar equivalent to the automaton has derivations that simulate the computations of the automaton: the grammar generates a phrase by means of a leftmost derivation if and only if the automaton executes a computation that recognises the same phrase. Therefore two different leftmost derivations correspond to two different computations.

PROPERTY – let M be a deterministic pushdown automaton; then the equivalent grammar G(M) is unambiguous.

NOTE: for simplicity, the general construction of the grammar equivalent to a deterministic automaton *with states* is not shown here; in the next lectures, however, it will be shown how to pass from a grammar to an equivalent det. automaton with states (the LL and LR methods); the general construction can be found in the literature (and is based on the intersection with REG).

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)