LANGAGES FORMELS

ENS Paris-Saclay

L3 informatique, 2025-2026

semaine 5

10. Grammars

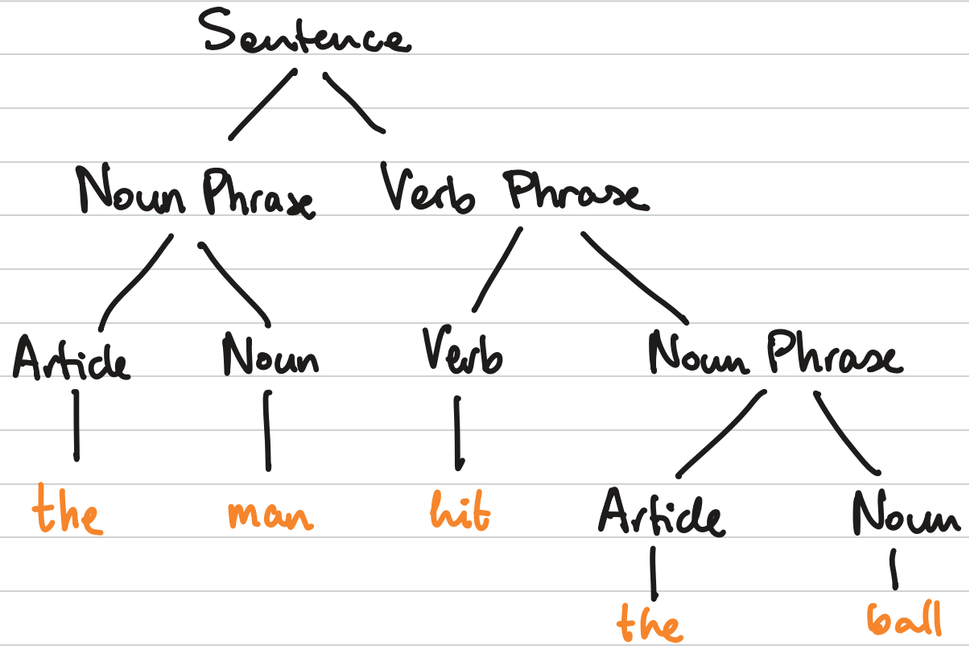
A generative approach to computation.

This chapter is partly based on the course of Jörg Endrullis: https://joerg.endrullis.de/automata/

The theory of formal grammar was developed in the 1950s by Chomsky and Schützenberger.

Initial motivations: connect linguistics to the theory of computation.

Example. (Chomsky 1957) The sentence "the man hit the ball" can be decomposed as:



The grammar rules.

$S \to NP \cdot VP$

$NP \to A \cdot N$

$VP \to V \cdot NP$

$A \to$ the

$N \to$ man | ball

$V \to$ hit

Abstracting away, we have: an alphabet V of variables (a.k.a. non-terminals),
an alphabet Σ of letters (a.k.a. terminals), } disjoint
an initial variable $S \in V$,
and a set of rules $\to \subseteq \Gamma^* V \Gamma^* \times \Gamma^*$,
where $\Gamma := V \cup \Sigma$.

A tuple $G = (V, \Sigma, S, \to)$ is called an unrestricted grammar or type 0 grammar.

Grammars generate languages in the alphabet Σ.

Let $G = (V, \Sigma, S, \to)$ be a grammar, and $\Gamma := V \cup \Sigma$.

Given words $w_1, w_2 \in \Gamma^*$, we write

$$w_1 \Rightarrow w_2$$

if there exists a rule $x_1 \to x_2$ in G and words $\alpha, \beta \in \Gamma^*$ such that $w_i = \alpha x_i \beta_i$.

We say that w_1 rewrites to w_2 in one step, in context α, β.

If $w_1 \Rightarrow^+ w_2$, we say w_1 rewrites to w_2. A $\Rightarrow$-path is called a derivation.

Define

$$\mathcal{L}(G) := \{ w \in \Sigma^* \mid S \Rightarrow^+ w \}.$$

Example. $S \to aAbc \mid abc$ notation for: $S \to aAbc$ and $S \to abc$ are rules

$A \to aAB \mid aB$ —"— $A \to aAB$ —"— $A \to AB$ —"—

$Bb \to bB$

$Bc \to bcc$

A derivation in this grammar:

$S \Rightarrow aAbc \Rightarrow aaABbc \Rightarrow aaAbBc \Rightarrow aaAbbcc$

$\Rightarrow aaaBbbcc \Rightarrow aaabBbcc \Rightarrow aaabbBcc \Rightarrow aaabbbccc$.

Exercise. Prove that $\mathcal{L}(G) = \{ a^n b^n c^n \mid n \geq 1 \}$.

The above grammar has the property that words never get shorter in a derivation.

A grammar G is **non-contracting** if, for every rule $w_1 \to w_2$, $|w_1| \leq |w_2|$.

A grammar G is **context-sensitive** if all of its rules are of the form

$$xAy \to xvy \quad \text{for } x, y \in \Gamma^*, A \in V, \text{ and } v \in \Gamma^+.$$

Proposition. A language $L \subseteq \Sigma^+$ is generated by a non-contracting grammar if, and only if, L —"— context-sensitive —"— .

Proof. $\Rightarrow$ First introduce a variable V_a for each $a \in \Sigma$, replace all letters by variables, and, for each $a \in \Sigma$, add the rule $V_a \to a$, which is context-sensitive.

Then, replace every $X_1 \dots X_m \to Y_1 \dots Y_n$ $(n \geq m)$ by $2m$ context-sensitive rules: $X_1 \dots X_m \to Z_1 X_2 \dots X_m$, $Z_1 X_2 \dots X_m \to Z_1 Z_2 X_3 \dots X_m$, ...,

$$Z_1 Z_2 \dots Z_{m-1} X_m \to Z_1 \dots Z_{m-1} Z_m Y_{m+1} \dots Y_n, \text{ and}$$

$$Z_1 \dots Z_m Y_{m+1} \dots Y_n \to Y_1 Z_2 \dots Z_m Y_{m+1} \dots Y_n, \dots, Y_1 Y_2 \dots Y_{m-1} Z_m Y_{m+1} \dots Y_n \to Y_1 \dots Y_m \dots Y_n.$$

$\Leftarrow$ Context-sensitive grammars are non-contracting. $\square$

Note. The variables $Z_1, \ldots, Z_m$ introduced in the proof should all be distinct from each other and also from $X_1, \ldots, X_m$ and $Y_1, \ldots, Y_n$.

For example, the non-contracting grammar

$$S \longrightarrow AB$$
$$AB \longrightarrow BC$$
$$BB \longrightarrow xx$$

clearly does not produce any words.

However, if we just replace the rule $AB \to BC$ by $AB \to BB$ and $BB \to BC$, then xx becomes derivable: $S \to AB \to BB \to xx$.

The new variables Z_i prevent this from happening: once we use a rule $X_1 \ldots X_n \to Z_1 X_2 \ldots X_n$, the only way to eliminate Z_1 is to at some point apply the remaining $2m-1$ rules to replace $Z_1 X_2 \ldots X_n$ by $Y_1 \ldots Y_m$.

Exercise. Prove more formally that the grammar given in the proof still defines the same language.

Example

$S \to aAbc \mid abc$

$A \to aAB \mid aB$

$Bb \to bB$

$Bc \to bcc$

replace letters by variables:

$S \to V_a A V_b V_c \mid V_a V_b V_c$

$A \to V_a AB \mid V_a B$

$BV_b \to V_b B$

$BV_c \to V_b V_c V_c$

$V_a \to a$, $V_b \to b$, $V_c \to c$

make rules context-sensitive, e.g.,

$BV_c \to V_b V_c V_c$ is replaced by: $BV_c \to Z_1 V_c$, $Z_1 V_c \to Z_1 Z_2 V_c$,

$Z_1 Z_2 V_c \to V_b Z_2 V_c$ and $V_b Z_2 V_c \to V_b V_c V_c$.

and similarly for the rule $BV_b \to V_b B$.

"contextuel"

A language $L \subseteq \Sigma^*$ is context-sensitive or type 1 if there exists a context-sensitive / non-contracting grammar such that $\mathcal{L}(G) = L - \{\varepsilon\}$.

↓

CSG can't generate ε :(

"hors contexte"

A grammar is context-free or type 2 if all rules are of the form

$$A \to w \quad \text{for } A \in V \text{ and } w \in \Gamma^*$$

A grammar is right-linear or type 3 if all rules are of the form

$$A \to aB \quad \text{or} \quad A \to a \quad \text{for } A, B \in V \text{ and } a \in \Sigma \cup \{\varepsilon\}.$$

We thus have four classes of languages (type 0 - 3).

What is their computational strength?

Type	Name	Expressive power
0	Unrestricted	Recursively enumerable (any TM)
1	Context-sensitive	Non-deterministic linear space TM ("LBA")
2	Context-free	Non-deterministic pushdown automaton
3	Right-linear	Finite automata

We will prove the correspondences 0, 1, 3 today. For 2, see second half of the course.

Example. Let $G = (\{a,b\}, \{S\}, \rightarrow, S)$, with rules:

- $S \rightarrow \varepsilon$
- $S \rightarrow a$
- $S \rightarrow b$
- $S \rightarrow aSa$
- $S \rightarrow bSa$

The grammar is of type 2 and not of type 3. (It's of type 1 if we remove the rule $S \rightarrow \varepsilon$.)

The language $\mathcal{L}(G) = \{ w \in \{a,b\}^* \mid \text{reverse}(w) = w \}$. (palindromes)

There does not exist a type 3 grammar that generates this language, since $\mathcal{L}(G)$ is not regular (by pumping argument).

Example In the context-free grammar G given by

$$S \to aA$$

$$A \to Sb$$

$$S \to \varepsilon$$

all of the rules are left- or right-linear.

However, $\mathcal{L}(G) = \{ a^n b^n : n \geq 0 \}$ is not regular, so G cannot be turned into a right-linear grammar.

Theorem. A language $L \subseteq \Sigma^*$ is regular if, and only if, $L = \mathcal{L}(G)$ for some right-linear grammar G.

Proof. $\Rightarrow$ Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA that recognizes L.

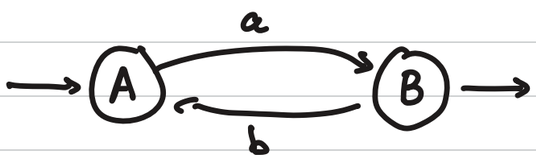
Define $G = (Q \cup \{S\}, \Sigma, \rightarrow, S)$ with rules

- $S \longrightarrow i$ for every $i \in I$
- $p \longrightarrow aq$ for every $(p, a, q) \in \delta$,
- $q \longrightarrow \varepsilon$ for every $q \in F$.

Then, for any $w \in \Sigma^+$, we have $i \Rightarrow^+ w$ if, and only if, A accepts w.

(Proof by induction on w.)

Example. The automaton



becomes the right-linear grammar:

$$S \to A$$

$$A \to aB$$

$$B \to bA$$

$$B \to \varepsilon \ .$$

Theorem. A language $L \subseteq \Sigma^*$ is regular if, and only if, $L = \mathcal{L}(G)$ for some right-linear grammar G.

Proof. $\Leftarrow$ Let $G = (V, \Sigma, S, \to)$ be right-linear with $\mathcal{L}(G) = L$.

Define $A := (V \sqcup \{H\}, \Sigma, S, \{H\}, \delta)$, where the transitions of δ are:

- $(A, a, B) \in \delta$ for every rule of the form $A \to aB$ in G
- $(A, a, H) \in \delta$ ——"—— $A \to a$ ——"——

Then A is an NFA with ε-transitions, and, for any word $w \in \Sigma^*$, we have

$S \Rightarrow^+ w$ if, and only if, A accepts w.

Thus, $L = \mathcal{L}(G) = \mathcal{L}(A)$ is regular. □

Example. The right-linear grammar

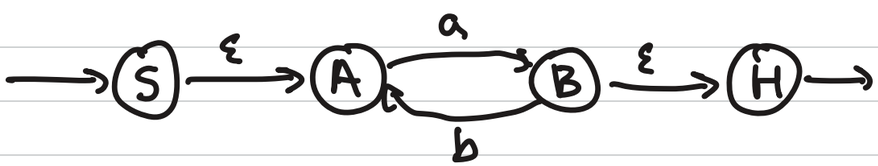
$$S \longrightarrow A$$

$$A \longrightarrow aB$$

$$B \longrightarrow bA$$

$$B \longrightarrow \varepsilon$$

gives the ε-NFA:



Theorem. A language $L \subseteq \Sigma^*$ is of type 0 if, and only if, L is recursively enumerable.

Proof. Type 0 $\Rightarrow$ r.e. Let $G = (\Sigma, V, \rightarrow, S)$ be a grammar such that $\mathcal{L}(G) = L$.

Define a non-deterministic Turing machine M with two tapes (input and work tape) and alphabet $V \cup \Sigma$, which does the following:

- Place S on the work tape;
- Repeat the following:
  - Guess a rule $w_1 \longrightarrow w_2$ of G;
  - Guess a factor w_1 of the work tape (if there is none, restart the loop);
  - Replace it by w_2 (increase or decrease spacing as needed);
  - Check if the work tape equals the input tape. If so, HALT.

The machine M halts on input w if, and only if, $w \in \mathcal{L}(G)$.

Remark If G is non-contracting, we can restrict the work tape to have length $|w|$, since any derivation of w can only contain words of length $\leq |w|$.

blank symbol $\notin \Sigma$

Proof. r.e. $\Rightarrow$ type 0. Let $M = (Q, \Sigma, \delta, q_0, \square, F)$ a non-deterministic Turing machine that accepts L.

We define a grammar $G = (V, \Sigma, S, \to)$ where

$$V := \{S, T\} \sqcup \left\{ \binom{a}{b} : a, b \in \Sigma \cup \{\square\} \right\} \sqcup \left\{ \binom{a}{qb} : a, b \in \Sigma \cup \{\square\}, q \in Q \right\}.$$

The idea is to simulate executions of M by rewrite rules of G. Three types of rules:

1) Guessing the input word:

$$S \to \binom{\square}{\square} S \mid S \binom{\square}{\square} \mid T$$

$$T \to T \binom{a}{a} \mid \binom{a}{q_0 a}$$

2) Simulating the execution of M:

for all $\alpha, \beta, \gamma \in \Sigma \cup \{\square\}$,

write d, move right, next state q'

$$\binom{\alpha}{qc}\binom{\beta}{\gamma} \to \binom{\alpha}{d}\binom{\beta}{q'\gamma} \quad \text{if} \quad \delta(q,c) = (q', d, R)$$

$$\binom{\beta}{\gamma}\binom{\alpha}{qc} \to \binom{\beta}{q'\gamma}\binom{\alpha}{d} \quad \text{if} \quad \delta(q,c) = (q', d, L)$$

3) Read the output when halted:

for all $\alpha, \beta, \gamma \in \Sigma \cup \{\square\}$,

$$\binom{\alpha}{q\gamma} \to \alpha \quad \text{if} \quad q \in F,$$

$$\beta \binom{\alpha}{\gamma} \to \beta\alpha, \quad \binom{\alpha}{\gamma}\beta \to \alpha\beta, \quad \square \to \varepsilon.$$

An accepting execution of M on input $a_1 \dots a_m \in \Sigma^*$ is simulated by a derivation in G as follows:

Rules

1) $S \to \begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix} S \mid S \begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix} \mid T$

$T \to T \begin{pmatrix}a\\ a\end{pmatrix} \mid \begin{pmatrix}a\\ q_0 a\end{pmatrix}$

2)

$\begin{pmatrix}\alpha\\ qc\end{pmatrix}\begin{pmatrix}\beta\\ \gamma\end{pmatrix} \to \begin{pmatrix}\alpha\\ d\end{pmatrix}\begin{pmatrix}\beta\\ q'\gamma\end{pmatrix}$ if $\delta(q,c) \ni (q', d, R)$

$\begin{pmatrix}\beta\\ \gamma\end{pmatrix}\begin{pmatrix}\alpha\\ qc\end{pmatrix} \to \begin{pmatrix}\beta\\ q'\gamma\end{pmatrix}\begin{pmatrix}\alpha\\ d\end{pmatrix}$ if $\delta(q,c) \ni (q', d, L)$

3) $\begin{pmatrix}\alpha\\ q\gamma\end{pmatrix} \to \alpha$ if $q \in F$, (*)

$\beta \begin{pmatrix}\alpha\\ \gamma\end{pmatrix} \to \beta\alpha$, $\begin{pmatrix}\alpha\\ \gamma\end{pmatrix}\beta \to \alpha\beta$, $\sqcup \to \varepsilon$.

Derivations

$S \Rightarrow^+ \begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix}^n \begin{pmatrix}a_1\\ q_0 a_1\end{pmatrix}\begin{pmatrix}a_2\\ a_2\end{pmatrix} \dots \begin{pmatrix}a_m\\ a_m\end{pmatrix}\begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix}^n$

$\Rightarrow^+$ simulate the execution of M in the bottom line, until M reaches a halting state:

$\begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix}^{n'} \begin{pmatrix}a_1\\ a_1\end{pmatrix} \dots \begin{pmatrix}a_k\\ q a_k\end{pmatrix} \dots \begin{pmatrix}a_m\\ a_m\end{pmatrix}\begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix}^{n''}$

$\overset{(*)}{\Rightarrow}$ $\begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix}^{n'} \begin{pmatrix}a_1\\ a_1\end{pmatrix} \dots \; a_k \; \dots \begin{pmatrix}a_m\\ a_m\end{pmatrix}\begin{pmatrix}\sqcup\\ \sqcup\end{pmatrix}^{n''}$

$\Rightarrow^*$ $a_1 \dots a_m$

Remark. If M only uses the space of the input, then we can omit all the rules containing $\sqcup$.

A **linear bounded automaton (LBA)** is a non-deterministic Turing machine M whose head never moves beyond the bounds of the input on the tape.

Equivalently, we can allow using $O(|w|)$ space for an input word w (increase the alphabet size).

Theorem. A language $L \subseteq \Sigma^+$ is generated by a context-sensitive grammar if, and only if, L is accepted by a linear bounded automaton.

Proof. By the remarks in the previous proof, and the fact that CS $\equiv$ non-contracting. □

Remark. · By Savitch's Theorem, $NSPACE(O(n)) \subseteq DSPACE(O(n^2))$.

· **Theorem (Immerman-Szelepcsényi, 1987)** $NSPACE(s(n))$ is closed under complement for any $s(n) \geq \log n$.

Therefore, context-sensitive languages are closed under complement.

Open Problem Are linear bounded automata determinizable, i.e., is

$$NSPACE(O(n)) \subseteq DSPACE(O(n)) \; ?$$

11. MSO

Describing regular languages with logic

We add to first order logic the possibility to quantify over subsets of a structure.

Formally, we have two sets of variables, V_1 and V_2, and corresponding quantifiers.

We also have two new kinds of atomic formulas:

- $X \subseteq Y$ for $X, Y \in V_2$
- $X(x)$ for $x \in V_1$ and $X \in V_2$.

The semantics in a structure $\mathcal{M}$ and valuation $v_1 : V_1 \to M$, $v_2 : V_2 \to \mathcal{P}(M)$ is:

$\mathcal{M}, v_1, v_2 \models X \subseteq Y$ iff $v_2(X)$ is a subset of $v_2(Y)$

$\mathcal{M}, v_1, v_2 \models X(x)$ iff $v_1(x)$ is an element of $v_2(X)$.

$\mathcal{M}, v_1, v_2 \models \forall X \varphi$ iff for all $S \in \mathcal{P}(M)$, $\mathcal{M}, v_1, v_2[X \mapsto S] \models \varphi$.

We consider monadic second order logic (MSO) over the signature $\{\leq^{(2)}\} \cup \{a^{(1)} : a \in \Sigma\}$, and we use finite words as structures (in the same way as we did for FO logic).

$w \mapsto \mathcal{M}_w$

Example. The set $\{ w \in \Sigma^* \mid w \text{ has even length} \}$ can be defined in MSO by:

$$\exists X \Big(\forall x \big((\mathit{first}(x) \to X(x)) \wedge (\mathit{last}(x) \to \neg X(x)) \wedge \forall y \, (\mathit{succ}(x,y) \to (X(x) \leftrightarrow \neg X(y))) \big) \Big),$$

where $\mathit{first}(x) := \forall y \, (x \leq y)$, $\mathit{last}(x) := \forall y \, (y \leq x)$,

$\mathit{succ}(x,y) := (x \leq y \wedge \forall z \, (x < z \to y \leq z))$.

The set $\{ a^n b^n \mid n \geq 0 \}$ is not definable in MSO.

Theorem. A language $L \subseteq \Sigma^*$ is MSO-definable if, and only if, L is regular.

For the proof of $\Rightarrow$, it is convenient to restrict the syntax of MSO to only use monadic second-order variables, and no FO variables.

For this, define

$$\text{empty}(X) := \forall Y\,(X \subseteq Y) \quad \text{and} \quad \text{sing}(X) := \neg\,\text{empty}(X) \wedge \forall Y\,(Y \subseteq X \rightarrow ((X \subseteq Y) \vee \text{empty}(Y)).$$

Then, in any structure $\mathcal{M}$, we have $\mathcal{M}, v_1, v_2 \models \text{sing}(X)$ iff $\#v_2(X) = 1$.

We further allow two new atomic formulas: $X \subseteq Y$ (meaning: for all $x \in X, y \in Y$, $x \leq y$)

and $a(X)$ (meaning: for all $x \in X$, $a(x)$).

Then in an MSO-formula φ, we can replace any occurrence of $x \in V_1$ by $X \in V_2$, and any quantifier $\forall x\,\psi$ by $\forall X\,(\text{sing}(X) \rightarrow \psi)$. This does not change the semantics of φ, and the resulting formula is pure MSO (no FO variables).

A formula φ of pure MSO on words over Σ, variables V, is inductively defined as:

- $X \leq Y$, $X \in Y$, $a(X)$ are formulas for $X, Y \in V$ and $a \in \Sigma$
- $\varphi \to \psi$ and $\bot$ are formulas if φ, ψ are formulas
- $\exists X \varphi$ is a formula.

Given $w \in \Sigma^*$ and $\upsilon : V \to \mathcal{P}(|w|)$, we denote by w^υ the word in $(\Sigma \times 2^V)^*$ which has the same length as w, and at position $0 \leq i < |w|$, has the letter $(w[i], \alpha_\upsilon)$ where $\alpha_\upsilon \in 2^V$ is defined by $\alpha_\upsilon(X) := \begin{cases} 1 & \text{if } i \in \upsilon(X) \\ 0 & \text{otherwise} \end{cases}$

Example. $w = aaba$, $\upsilon : X \mapsto \{0,1\}$, $Y \mapsto \{1,2\}$. Then

$$w^\upsilon = \begin{array}{cccc} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ a & a & b & a \end{array} \quad \begin{array}{l} \leftarrow Y \\ \leftarrow X \end{array}$$

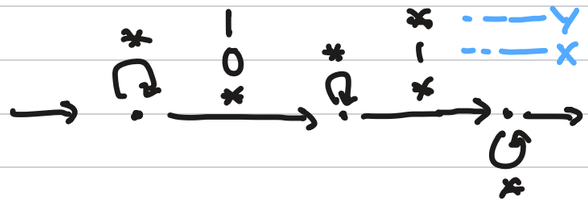
For a pure MSO-formula φ, if $V := FV(\varphi)$, we define

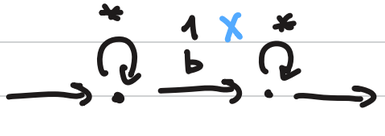
$$\mathcal{L}(\varphi) := \{ w^\upsilon \mid w \in \Sigma^*, \upsilon : V \to \mathcal{P}(|w|) \text{ such that } \mathcal{M}_{w,\upsilon} \models \varphi \},$$

a language in alphabet $\Sigma \times 2^V$.

Theorem. A language $L \subseteq \Sigma^*$ is MSO-definable if, and only if, L is regular.

Proof of $\Rightarrow$ By induction on a pure MSO-formula φ, we will show that $\mathcal{L}(\varphi)$ is regular.

- $X \leq Y$: The NFA  recognizes $\mathcal{L}(X \leq Y)^c$.

- $a(X)$:  recognizes $\mathcal{L}(a(X))^c$.

- $\rightarrow, \perp$: Rec is closed under union and complement

- $\exists X \varphi$: If X does not occur in φ, then $\mathcal{L}(\exists X \varphi) = \mathcal{L}(\varphi) - \{\varepsilon\}$, which is still regular.

  Assume X in φ. By IH, let A an automaton over the alphabet $\Sigma \times 2^{V \cup \{X\}}$, with $\mathcal{L}(\varphi) = \mathcal{L}(A)$.

  Define A' over $\Sigma \times 2^V$ by erasing the bit for X.

  Then A' accepts w^v iff there exists an extension v' of v to the variable X such that A accepts $w^{v'}$. □

Proof of $\Leftarrow$. Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA, with $Q = \{1, \dots, n\}$.

Idea: Guess a successful run, use FO logic to check that it is correct.

Formally, for the guess, use n MSO-variables $X_1, \dots, X_n$ and the following formulas:

$$\text{init} := \forall x \Big(\text{first}(x) \to \bigvee_{i \in I} \bigvee_{a \in \Sigma} \bigvee_{j : (i,a,j) \in \delta} \big(a(x) \wedge X_j(x) \big) \Big),$$

$$\text{final} := \forall x \Big(\text{last}(x) \to \bigvee_{i \in F} X_i(x) \Big),$$

$$\text{trans} := \forall x \forall y \Big(\text{succ}(x,y) \to \bigvee_{i=1}^{n} \bigvee_{j=1}^{n} \bigvee_{a \in \Sigma : (i,a,j) \in \delta} \big(a(x) \wedge X_i(x) \wedge X_j(y) \big) \Big),$$

$$\text{part} := \forall x \Big(\bigvee_{i=1}^{n} \big(X_i(x) \wedge \bigwedge_{j \neq i} \neg X_j(x) \big) \Big).$$

For w a word and $\upsilon_2 : \{X_1, \dots, X_n\} \to \mathcal{P}(|w|)$, we have $\mathcal{M}_w, \upsilon_2 \models \text{init} \wedge \text{final} \wedge \text{trans} \wedge \text{part}$ iff $(\upsilon_2 X_1, \dots, \upsilon_2 X_n)$ labels a successful run of A on w.

So $\varphi_A := \exists X_1 \dots \exists X_n \, (\text{init} \wedge \text{final} \wedge \text{trans} \wedge \text{part})$ is an MSO-sentence defining $\mathcal{L}(A)$. $\square$

Corollary. MSO is decidable over finite linear orders.

Proof. Given an MSO formula φ, it is satisfiable iff the language $\mathcal{L}(\varphi)$ is non-empty. This happens iff the constructed automaton accepts some word. □

The above technique can be extended to MSO on other structures, such as infinite words and trees, and lies at the heart of many verification algorithms.

→ see Logique course and M1 courses on Verification and Model Checking.

Bonnes vacances !

Theo Jansen, Animaris Suspendisse (2016)

https://www.kqed.org/arts/11633101/giant-sand-dwelling-skeletons-descend-on-the-exploratorium