Selected Topics in Automata and Logic

Rafael Peñaloza

SS 2010

Contents

1 Regular Languages and Logic

We first recall some of the basic notions and results of regular languages and automata theory. We then investigate their relation to (extensions of) first-order logic.

1.1 Regular Languages and Finite Automata

Definition 1.1. Let Σ be a finite *alphabet*. The class $\mathbf{REG}_\Sigma$ of *regular languages* over Σ is the smallest class such that:

- $\emptyset, \{\varepsilon\}$, and $\{a\}$ are in $\mathbf{REG}_\Sigma$ for all $a \in \Sigma$ (where ε is the empty word)
- if $L_1, L_2 \in \mathbf{REG}_\Sigma$ then so are $L_1 \cup L_2$, $L_1 \cdot L_2 = \{u \cdot v \mid u \in L_1, v \in L_2\}$, and $L_1^* = \{u_1 \cdots u_n \mid n \geq 0, u_i \in L_1\}$.

Regular languages are usually described by *regular expressions*. For example, the expression ab^*a describes $\{a\} \cdot \{b\}^* \cdot \{a\}$; that is, all words starting and ending with a, and having only b's inbetween.

A different characterization of regular languages can be made via finite automata.

Definition 1.2. A non-deterministic *finite automaton* is a tuple of the form $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, where

- Q is a finite set of *states*,
- Σ is a finite alphabet,
- $I \subseteq Q$ is the set of *initial states*,
- $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*, and
- $F \subseteq Q$ is the set of *final states.*

One can use graphs to represent automata. Nodes represent the states and (labeled) edges represent the transition relation. Special markings are used to show the initial and final states. For example, the graph

$$\begin{tikzcd}
{} \arrow[r] & \textcircled{1} \arrow[r, bend left, "a"] & \textcircled{\textcircled{2}} \arrow[l, bend left, "b"]
\end{tikzcd}$$

represents the automaton with $Q = \{1, 2\}$, $\Delta = \{(1, a, 2), (2, b, 1)\}$, $I = \{1\}$ (represented by $\rightarrow\textcircled{1}$), and $F = \{2\}$ (represented by $\textcircled{\textcircled{2}}$).

A *path* in the automaton is a sequence $q_0 a_1 q_1 a_2 \ldots a_n q_n$ where for every $i, 1 \leq i \leq n, (q_{i-1}, a_i, q_i) \in \Delta$. Such a path is *successful* if $q_0 \in I$ and $q_n \in F$. The automaton *accepts* a word w if there is a successful path such that $w = a_1 a_2 \ldots a_n$. The language accepted by the automaton $\mathcal{A}$ is $L(\mathcal{A}) = \{w \in \Sigma^* \mid w \text{ is accepted by } \mathcal{A}\}$.

A language $L \subseteq \Sigma^*$ is called *recognizable* if there exists a finite automaton $\mathcal{A}$ that accepts L.

Kleene's Theorem says that a language $L \subseteq \Sigma^*$ is recognizable iff it is regular. This means that finite automata characterize the set of all recognizable languages. In fact, this characterization still holds, even if we restrict to only deterministic automata.

Definition 1.3. An automaton $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ is called *deterministic* iff

- $|I| = 1$ (i.e. $I = \{q_0\}$),
- Δ is functional; that is, for every $q \in Q$ and $a \in \Sigma$ there is **at most one** $q' \in Q$ such that $(q, a, q') \in \Delta$.

Note. Sometimes deterministic automata are required to have a transition for each $(q, a) \in Q \times \Sigma$. This does not make a difference with respect to expressivity, but our definition allows a more compact representation of the automata.

Given a non-deterministic automaton $\mathcal{A}$, one can construct (using the well-know power-set construction) a deterministic automaton $\mathcal{A}'$ that accepts the same language. This shows that deterministic automata are as expressive as non-deterministic ones.

Finally, recall that regular (and thus also recognizable) languages are closed with respect to the following operations:

- union ($\cup$), intersection ($\cap$),
- concatenation ($\cdot$),
- Kleene star (*), and
- complement ($^-$).

1.2 First-Order- and Monadic-Second-Order-Logic

We now look at the representation of regular languages by logical formulae. We first recall how first-order formulae can be used to represent languages, and then extend it to monadic second-order logic, capable of expressing all regular languages.

We first assume w.l.o.g. that the alphabet Σ consists of binary tuples; that is, $\Sigma = \{0,1\}^k$ for some k. Notice that it is always possible to encode a given finite alphabet in such a way.

We now considere first-order predicate logic extended with the following symbols:

$$\underset{\text{binary}}{=}\;,\qquad \underset{\text{binary}}{<}\;,\qquad \underset{\text{unary symbols}}{P_1,\dots,P_k}$$

For the semantics, we consider only finite interpretations; w.l.o.g. we assume then that the domain of a given interpretation I is $\mathsf{dom}(I) = \{1,\dots,n\}$. Then, $=$ and $<$ are interpreted as the usual equality and order of natural numbers, and each P_j is interpreted as a set $P_j^I \subseteq \mathsf{dom}(I)$.

Define, for each $i, 1 \leq i \leq n$ the tuple $\sigma_i = (b_{i1},\dots,b_{ik})$, where

$$b_{ij} = \begin{cases} 1 & \text{if } i \in P_j^I, \\ 0 & \text{if } i \notin P_j^I. \end{cases}$$

Then, the interpretation I corresponds to the word $\sigma_1\sigma_2\dots\sigma_n$. Conversely, it is easy to see that for each such word, one can build a (unique) interpretation I.

Example 1.4. Let $k = 2$ (that is, $\Sigma = \{0,1\}^2$). The word

$$\binom{1}{0}\binom{1}{1}\binom{0}{0}\binom{1}{0}$$

over Σ corresponds to the interpretation I where

- $\mathsf{dom}(I) = \{1,2,3,4\}$,
- $P_1^I = \{1,2,4\}$, and
- $P_2^I = \{2\}$.

Given this equivalence between words and interpretations, it makes sense to say that a word $w \in \Sigma^+$ makes a formula φ true ($w \models \varphi$).

Definition 1.5. Let $\Sigma = \{0,1\}^k$, and φ be a closed formula (i.e., without free variables) of first-order predicate logic using the extra-logical symbols $=, <, P_1,\dots,P_k$. Then φ defines the language

$$L(\varphi) = \{w \in \Sigma^+ \mid w \models \varphi\}.$$

Note. Interpretations cannot have empty domains. Thus, the empty word does not describe any interpretation; i.e., $\varepsilon \notin L(\varphi)$. However, this is not a strong restriction since, for instance, a language $L \subseteq \Sigma^*$ is regular iff $L \setminus \{\varepsilon\}$ is regular.

Example 1.6. Let $k = 1$. The language 10^*1 is defined by the formula:

$$\begin{aligned}&\exists x,y.(P_1(x) \land P_1(y) \land x < y \land \\ &\qquad \forall z.(\neg(z < x) \land \neg(y < z) \land (x < z < y \Rightarrow \neg P_1(z)))).\end{aligned}$$

Since we are interested in languages, we introduce the following useful abbreviations:

- $Q_\sigma(x)$: for every $\sigma \in \Sigma$, the formula $Q_\sigma(x)$ with one free variable that expresses that σ occurs at position x. For instance, $Q_{(0,1)}(x) := \neg P_1(x) \land P_2(x)$.
- $\mathsf{Min}(x)$: this formula says that x is the first position of the word: $\mathsf{Min}(x) := \neg\exists y.(y < x)$. We sometimes use simply the constant min to represent this position; it is always interpreted as 1.

- $\mathsf{Max}(x)$: expresses the last position of the word: $\mathsf{Max}(x) := \forall y.(y \leq x)$. The constant max is also used.
- $\mathsf{Succ}(x, y)$: y is the successor position of x: $\mathsf{Succ}(x, y) := x < y \wedge \neg\exists z.(x < z \wedge z < y)$. The function symbol $\mathsf{s}(x)$ is also used.

Example 1.7. The language ab^*a is defined by

$$Q_a(\min) \wedge Q_a(\max) \wedge \min \neq \max \wedge \\ \forall x.(\min < x < \max \Rightarrow Q_b(x)).$$

However, first-order logic does not suffice for expressing all regular languages. For example, there is no first-order formula that can express the language $a(aa)^*$, which is obviously regular. In fact, it can be shown that first-order logic can only express *star-free languages*.

Definition 1.8. Let Σ be a finite alphabet. The class $\mathbf{SF}_\Sigma$ of all *star-free languages* over Σ is the smallest class such that:

- all finite languages over Σ belong to $\mathbf{SF}_\Sigma$,
- if $L_1, L_2 \in \mathbf{SF}_\Sigma$, then so are $L_1 \cdot L_2$, $L_1 \cup L_2$, $L_1 \cap L_2$ and $\overline{L} = \Sigma^* \setminus L$.

Example 1.9. For $\Sigma = \{a, b\}$, the language $ab^*a \in \mathbf{SF}_\Sigma$ since

$$ab^*a = a \cdot (\overline{\Sigma^* \cdot a \cdot \Sigma^*}) \cdot a$$

and $\Sigma^* = \overline{\emptyset}$.

Proposition 1.10. *For a language $L \subseteq \Sigma^+$ the following are equivalent:*

1. *$L \in \mathbf{SF}_\Sigma$.*
2. *there is a closed first-order formula φ with $L = L(\varphi)$.*

In order to capture the whole expressivity of regular languages, we allow second-order quantifiers $\forall X, \exists X$ ranging over subsets of the domain. This logic is called S1S.

Example 1.11. The language $a(aa)^*$ is defined by the formula

$$\forall x Q_a(x) \wedge \exists X \exists Y (X(\min) \wedge X(\max) \wedge \forall x(X(x) \Leftrightarrow \neg Y(x)) \wedge \\ \forall x, y(X(x) \wedge \mathsf{Succ}(x, y) \Rightarrow Y(y)) \wedge \\ \forall x, y(Y(x) \wedge \mathsf{Succ}(x, y) \Rightarrow X(y))).$$

Proposition 1.12. *For a language $L \subseteq \Sigma^+$ the following are equivalent:*

1. *$L \in \mathbf{REG}_\Sigma$.*
2. *there is a closed S1S formula φ with $L = L(\varphi)$.*

2 Other Equivalent Representations

In this chapter we introduce a new automata model and a new (first-order) logic, that can express exactly the class of regular languages.

2.1 Two-way Finite Automata

We now introduce a generalization of finite automata, which is able to move backwards while processing a word. These are called two-way finite automata.

In order to simplify the constructions and proofs, we will assume that the exclusive symbols $\triangleright$ and $\triangleleft$ appear at the beginning and the end of each input word, respectively. For instance, the word aa, will be represented by the chain $\triangleright aa \triangleleft$.

Note. The language $L \subseteq \Sigma^*$ is accepted by a (one-way) finite automaton iff there is a finite automaton that accepts $\triangleright L \triangleleft$. (EXERCISE!)

Definition 2.1. A *two-way finite automaton* is a tuple of the form $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, where

- Q is a finite set of *states*,
- Σ is a finite alphabet,
- $I \subseteq Q$ is the set of *initial states*,
- $\Delta \subseteq Q \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times Q \times \{-1, 0, 1\}$ is the *transition relation*, and
- $F \subseteq Q$ is the set of *final states.*

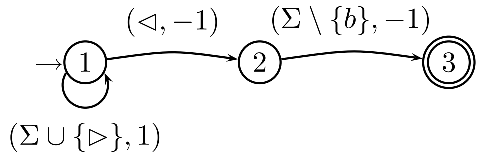
Two-way finite automata only differ from finite automata as defined in the previous chapter in the transition relation Δ. Intuitively, a transition does not only express the new state of the automaton, but also whether the head should move to the left (-1), right (1) or stay in the same place (0). We assume that if the head reads the symbol $\triangleright$, then there is no transition that moves the head to the left. Likewise, no transition moves the head to the right if the head is reading $\triangleleft$.

The notion of *deterministic* two-way automata is analogous to the one-way case; that is, a two-way automaton is deterministic if it has only one initial state and there is at most one transition for any given state and symbol in $\Sigma \cup \{\triangleright, \triangleleft\}$.

A *configuration* of a two-way automaton $\mathcal{A}$ is an element of $Q \times \mathbb{N}$, where the first component indicates the state of the automaton, and the second the *position* of the head. A *run* is a sequence of configurations (or, equivalently, a word in $(Q \times \mathbb{N})^*$). The sequence $(q_0, j_0) \dots (q_m, j_m)$ is a run of $\mathcal{A}$ on a word $w = a_1 \dots a_n \in \Sigma^*$ if $q_0 \in I, j_0 = 0$, and for all $i, 0 \leq i \leq m$ it holds: (i) $0 \leq j_i \leq n+1$ and (ii) $(q_i, a_{j_i}, q_{i+1}, j_{i+1} - j_i) \in \Delta$, where $a_0 = \triangleright$ and $a_{n+1} = \triangleleft$. This run is *accepting* if $q_m \in F$. The automaton $\mathcal{A}$ accepts w if it has an accepting run on w.

Two-way finite automata can also be represented using graphs. The only difference is that the edges (that represent the transition relation) are labeled with tuples $(\sigma, j) \in \Sigma \times \{-1, 0, 1\}$.

Example 2.2. The (deterministic) two-way automaton



$(\triangleleft, -1)$
$(\Sigma \setminus \{b\}, -1)$
1
2
3
$(\Sigma \cup \{\triangleright\}, 1)$

accepts the language $\Sigma^* \setminus (\Sigma^* b)$ for any alphabet Σ.

Clearly, two-way automata are more general than their one-way counterparts: every one-way finite automaton can be seen as a two-way automaton where all the transitions move the head to the right. As we will show next, they have in fact the same expressive power. That is, for every two-way automaton $\mathcal{A}$, we can construct a one-way automaton $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$. In particular, this means that two-way finite automata accept the class of regular languages.

To prove the equivalence of one-way and two-way automata, we first give a characterization of when a two-way automaton does *not* accept a word w.

Lemma 2.3. *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a two-way automaton and $w = a_1 \dots a_n \in \Sigma^*$. $\mathcal{A}$ does not accept w if and only if there is a sequence $T_0, \dots, T_{n+1}$ of subsets of Q such that the following three conditions hold:*

1. *$I \subseteq T_0$,*
2. *$T_i \cap F = \emptyset$ for all $0 \leq i \leq n+1$, and*
3. *for $0 \leq i \leq n$, if $q \in T_i$ and $(q, a_i, q', k) \in \Delta$, then $q' \in T_{i+k}$.*

Proof. Suppose first that $\mathcal{A}$ does not accept w. Then we define, for each $0 \leq i \leq n+1$, the set T_i as the set of all states $q \in Q$ such that (q, i) is a configuration in a run of $\mathcal{A}$ in w. We show now that $T_0, \dots, T_{n+1}$ satisfies the three conditions of the lemma. If $q \in I$, then $(q, 0)$ is a run of $\mathcal{A}$ on w, and hence $I \subseteq T_0$. Since $\mathcal{A}$ does not accept w, then for every $q \in F$ it follows that no configuration of the form (q, i), with $0 \leq i \leq n+1$, occurs in any run of $\mathcal{A}$ on w; this means that $T_i \cap F = \emptyset$ for all $0 \leq i \leq n+1$. Finally, if $0 \leq i \leq n+1$ and $q \in T_i$, then there is a run $(q_0, j_0) \dots (q_l, j_l)$ of $\mathcal{A}$ on w such that $q_l = q$ and $j_l = i$. If $(q, a_i, q', k) \in \Delta$, then $(q_0, j_0) \dots (q_l, j_l)(q', i+k)$ is also a run of $\mathcal{A}$ on w, and thus $q' \in T_{i+k}$.

For the other direction, suppose now that $\mathcal{A}$ accepts w. Then there must be an accepting run of $\mathcal{A}$ on w. Let $(q_0, j_0) \dots (q_m, j_m)$ be such a run. In particular, it follows that $q_m \in F$. We show now that no sequence $T_0, \dots, T_{n+1}$ can satisfy the three conditions of the lemma. Let $T_0, \dots, T_{n+1}$ be a sequence that satisfies the first and third conditions. We show by induction on i that $q_i \in T_{j_i}$ for $0 \leq i \leq m$. Clearly, $q_0 \in T_0$ because $q_0 \in I$ and $I \subseteq T_0$. For the induction step, assume that we have shown that $q_i \in T_{j_i}$ for some $0 \leq i < m$. Then there is a transition $(q_i, a_{j_i}, q_{i+1}, k) \in \Delta$ such that $j_{i+1} = j_i + k$. But then, $q_{i+1} \in T_{j_{i+1}}$. This in particular shows that $q_m \in T_{j_m}$, but then $T_{j_m} \cap F \neq \emptyset$, and hence the sequence cannot satisfy the second condition of the lemma. □

With the help of this lemma we can now show equivalence of two-way and one-way automata.

Theorem 2.4. *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a two-way automaton. Then there is a one-way automaton $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$.*

Proof. We prove this in an indirect way using the closure properties of one-way automata: instead of constructing a one-way automaton that accepts the language $L(\mathcal{A})$, we construct one that accepts the complement language $\Sigma^* \setminus L(\mathcal{A})$. However, we know that the class of regular languages is closed under complement, and so this means that there is an automaton that accepts $L(\mathcal{A})$.

Let $\mathcal{B}$ be the automaton $(P, \Sigma, J, \Lambda, G)$ where:

- $P = 2^{Q \setminus F} \cup (2^{Q \setminus F} \times 2^{Q \setminus F})$; that is, sets of non-final states and pairs of sets of non-final states from $\mathcal{A}$,
- $J = \{T \subseteq Q \setminus F \mid I \subseteq T\}$,
- $(T, \rhd, (T, U)) \in \Lambda$ if
  - if $q \in T$ and $(q, \rhd, q', 0) \in \Delta$, then $q' \in T$, and
  - if $q \in T$ and $(q, \rhd, q', 1) \in \Delta$, then $q' \in U$;
- $((T, U), a, (U, V)) \in \Lambda$ if
  - if $q \in U$ and $(q, a, q', -1) \in \Delta$, then $q' \in T$,
  - if $q \in U$ and $(q, a, q', 0) \in \Delta$, then $q' \in U$, and
  - if $q \in U$ and $(q, a, q', 1) \in \Delta$, then $q' \in V$.
- $(T, U) \in G$ if
  - if $q \in U$ and $(q, \lhd, q', -1) \in \Delta$, then $q' \in T$, and
  - if $q \in U$ and $(q, \lhd, q', 0) \in \Delta$, then $q' \in U$;

We show that $\mathcal{B}$ accepts a word $\rhd w$ iff $\mathcal{A}$ rejects w.

Assume first that $\mathcal{B}$ accepts $\rhd w$, where $w = a_1 a_2 \dots a_n$; then there is a succesful run of $\mathcal{B}$ of the form $T_0 \rhd (T_0, T_1) a_1 (T_1, T_2) a_2 \dots a_n (T_n, T_{n+1})$ with $(T_n, T_{n+1}) \in G$. We show that the sets $T_0, \dots, T_{n+1}$ satisfy the conditions of Lemma 2.3. 1. since $T_0 \in J$, then $I \subseteq T_0$. 2. since the sets that form the states cannot include any final states from $\mathcal{A}$, it follows that $T_i \cap F = \emptyset$ for all $i, 0 \leq i \leq n+1$. Finally, for 3. let $q \in T_i$ and $(q, a_i, q', k) \in \Delta$. If $i = 0$, then $a_0 = \rhd$ and the transition $(T_0, \rhd, (T_0, T_1)) \in \Lambda$ ensures $q' \in T_{i+k}$. If $i = n+1$, then the acceptance condition ensures that $q' \in T_{i+k}$. Otherwise, we have a transition $((T_{i-1}, T_i), a, (T_i, T_{i+1})) \in \Lambda$, and by construction it satisfies that $q' \in T_{i+k}$. Hence, $\mathcal{A}$ rejects w.

Assume now that $\mathcal{A}$ reject w. Then, there are sets $T_0, T_1, \ldots, T_{n+1}$ satisfying the conditions of Lemma 2.3. Then the run

$$T_0 \rhd (T_0, T_1)a_1(T_1, T_2)a_2 \ldots a_n(T_n, T_{n+1})$$

is successful for $\mathcal{B}$:

- $T_0 \in J$ because $I \subseteq T_0$;
- $T_0, (T_i, T_{i+1}) \in P$ for all $i, 0 \leq i \leq n$ since $T_i \cap F = \emptyset$,
- the transitions and the final state condition are guaranteed by Condition 3.

Hence $\mathcal{B}$ accepts $\rhd w$. $\square$

Obviously, this theorem also implies that deterministic and non-deterministic two-way finite automata accept the same class of languages: regular languages.

Example 2.5. Let $\Sigma = \{a\}$ and $L = a^+$. (Hand-written example)

2.2 First Order Logic with Transitive Closure

We now look at a logical representation that does not need (explicit) second-order constructors. What we use is the *transitive closure* operator.

Consider a formula $\varphi(x, y)$ with two free variables (x and y). We can also see this formula as a binary relation: $x\varphi y$ iff $\varphi(x, y)$ is true. We can thus also consider the reflexive, transitive closure of this relation. We denote this closure as $\mathsf{TC}[\varphi]$. Notice that $\mathsf{TC}[\varphi]$ still has two free variables.

The semantics of this operator is as follows: given two elements d, e of the domain $\mathsf{dom}(I)$, the interpretation I satisfies $\mathsf{TC}[\varphi(x, y)](d, e)$ if $d = e$ or there exist $d_1, \ldots, d_m \in \mathsf{dom}(I)$ such that

- $d = d_1$;
- $e = d_m$; and
- $I \models \varphi(d_i, d_{i+1})$ for all $1 \leq i < m$.

Intuitively, the transitive closure operator allows us to represent the Kleene star, which is the only missing piece of expressivity necessary for obtaining $\mathbf{REG}_\Sigma$.

Example 2.6. Consider the formula

$$\varphi(x, y) = s(x, y) \land Q_b(y),$$

and let

$$\psi = Q_a(\min) \land \mathsf{TC}[\varphi](\min, \max).$$

Then $L(\psi) = ab^*$.

We consider also a notion of *deterministic* transitive closure, where we add the restriction that if $\varphi(d, e_1)$ and $\varphi(d, e_2)$ hold, then $e_1 = e_2$. Formally, the operator DTC is defined as:

$$\mathsf{DTC}[\varphi(x, y)] = \mathsf{TC}[\varphi(x, y) \land \forall z.\varphi(x, z) \Rightarrow y = z].$$

Example 2.7. Let $\Sigma = \{a, b\}$, and consider the following formulas:

$$\begin{aligned} \varphi(x, y) &= Q_a(y), \\ \psi &= Q_a(\min) \land \mathsf{DTC}[\varphi](\max, \min). \end{aligned}$$

Then $L(\psi) = ab^*$.

Note. We could have changed TC to DTC in Example 2.6 and obtain the same language. However, changing DTC to TC in Example 2.7 would yield a different language, namely $a\Sigma^*$.

Theorem 2.8. *First-order logic extended with* TC *or* DTC *captures exactly the class of regular languages.*

Proof. We first show that FOL with DTC can express all regular languages. For this, we construct a formula that describes the behaviour of a deterministic automaton. Let $\mathcal{A} = (Q, \Sigma, \{q_0\}, \Delta, F)$ be a deterministic automaton and assume w.l.o.g. that $Q = \{1, \ldots, m\}$; additionally, we assume that there are no transitions from final states. We build recursively the formulas χ^n_{ij} whose intuitive meaning is as follows: $\chi^n_{ij}(p_1, p_2)$ is true iff whem $\mathcal{A}$ is started in state i with its head in position p_1, then $\mathcal{A}$ can move to state j in position p_2, while not passing through any state higher than n. For the base case, we set

$$\chi^0_{ij}(p_1, p_2) = \bigvee_{a\in\Sigma,(i,a,j)\in\Delta} Q_a(p_1) \wedge p_2 = s(p_1).$$

This formula is true iff there is a transition from state i to state j. For constructing the formula χ^n_{ij}, with $n > 0$, we need to be able to express that we can reach state j from state i without using any state greater than n. This can be decomposed as follows: first, we reach state n from i using no state higher than $n-1$ (i.e. χ^{n-1}_{in}), then we can reach as many times as we want the state n, using again only states smaller than n (i.e. χ^{n-1}_{nn}), and finally, reach state j from n (i.e. χ^{n-1}_{nj}). The second part of this intuitive construction ("as many times as we want") resembles a transitive closure. Furthermore, since $\mathcal{A}$ is deterministic, given an input word there is at most one position p_2 that satisfies $\chi^{n-1}_{nn}(p_1, p_2)$ for any given p_1. Hence, we can use the deterministic TC operator:

$$\begin{aligned}\chi^n_{ij}(p_1, p_2) = &\exists p_3, p_4.(\chi^{n-1}_{in}(p_1, p_3)\wedge\\ &\qquad \mathsf{DTC}[\chi^{n-1}_{nn}](p_3, p_4)\wedge\\ &\qquad \chi^{n-1}_{nj}(p_4, p_2))\vee\\ &\chi^{n-1}_{ij}(p_1, p_2).\end{aligned}$$

By definition, we know that DTC can be rewritten to TC, and hence also TC can express all regular languages. Now, to show that TC cannot express anything beyond regular languages, we show how to encode the TC operator in monadic second order logic. Let $\psi = \mathsf{TC}[\varphi](x, y)$. We define a second order formula $\theta(x)$ that expresses that A is the minimal set that contains x and is closed under φ:

$$\begin{aligned}\theta(x) =&(A(x) \wedge \forall v, w.A(v) \wedge \varphi(v, w) \Rightarrow A(w))\wedge\\ &(\forall B.(B(x) \wedge (\forall v, w.B(v) \wedge \varphi(v, w) \Rightarrow B(w))) \Rightarrow\\ &\quad(\forall v.A(v) \Rightarrow B(v))).\end{aligned}$$

The formula ψ can be substituted by $\exists A.\theta \wedge A(y)$ yielding an S1S formula. Thus, FOL with TC can only express regular languages. $\square$

3 Two-Way Multihead Automata

We now go beyond the realm of regular languages by allowing automata to read several symbols at the same time. That is, the automaton has several *heads* that traverse the input, and the internal state and motion of these heads depends on the symbols read by all of them.

Definition 3.1. A *two-way automaton with k heads* is a tuple of the form $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, where Q, Σ, I and F are as in regular two-way automata, and $\Delta \subseteq Q \times (\Sigma \cup \{\triangleright, \triangleleft\})^k \times Q \times \{-1, 0, 1\}^k$ is the transition relation.

Such an automaton is called *one-way* if no transition moves any head to the left; i.e., if $\Delta \subseteq Q \times (\Sigma \cup \{\triangleright, \triangleleft\})^k \times Q \times \{0, 1\}^k$. The notions of deterministic k-head automata are analogous as in the one-head case.

A configuration of a two-way k-head automaton $\mathcal{A}$ is an element of $Q \times \mathbb{N}^k$, where the first component indicates the state of the automaton and the second the position of each of the heads. A *run* is a sequence of configurations. The sequence $(q_0, j_{01}, \ldots, j_{0k}) \ldots (q_m, j_{m1}, \ldots, j_{mk})$ is a run of $\mathcal{A}$ on a word $w = a_1 \ldots a_n \in \Sigma^*$ if $q_0 \in I$, for all $1 \leq \ell \leq k$, $j_{0\ell} = 0$ and for all $i, 0 \leq i \leq m$ it holds that $0 \leq j_{i\ell} \leq n + 1$ and $(q_i, (a_{j_{i1}}, \ldots, a_{j_{ik}}), (j_{i+1,1} - j_{i1}, \ldots, j_{i+1,k} - j_{ik})) \in \Delta$. The run is accepting if $q_m \in F$.

Multi-head automata are strictly more expressive than one-head automata. For instance, the language $a^n b^n$ is not regular, but can be accepted by a deterministic one-way 2-head automaton (Exercise!).

In fact, even if we consider only one-way automata, adding more heads produces always an increase in expressivity. Before showing this, and other results, let us look at some examples of languages accepted by multi-head automata.

Example 3.2. Let $m \in \mathbb{N}$ and $\Sigma = \{0, 1, *\}$. Define the language

$$L_m = \{w_1 * w_2 * \ldots * w_{2m} \mid w_i \in \{0,1\}^*, w_i = w_{2m+1-i}, 1 \leq i \leq 2m\}.$$

In other words, L_m is the language of all words of the form

$$w_1 * \ldots * w_m * w_m * \ldots * w_1$$

where $w_i \in \{0,1\}^*, 1 \leq i \leq m$.

This language can be accepted by a deterministic automaton that has "enough" heads; more precisely,

Lemma 3.3. *For all $k, m \in \mathbb{N}$ such that $m \leq \binom{k}{2}$, there is a 1-kDFA that accepts L_m.*

Proof Sketch. The automaton that accepts L_m works as follows: the first head traverses the last $k-1$ words $w_{2m+2-k}, \ldots, w_{2m}$, while the remaining $k-1$ heads are used to compare these words with $w_{k-1}, \ldots, w_1$, respectively. Afterwards, the remaining $k-1$ heads are positioned at the beginning of word w_k and the same procedure can be inductively repeated to verify that $w_k * \ldots * w_{2m+1-k}$ is in L_{m+1-k}.

Note that: (i) the procedure is deterministic and (ii) the first head cannot be used in the recursive call, since it has already reached the end of the word. □

In fact, this number of heads is necessary, in the sense that there is no (non-deterministic) 1-kFA that accepts L_m if $m > \binom{k}{2}$.

Lemma 3.4. *For all $k, m \in \mathbb{N}$, if $m > \binom{k}{2}$, then there is no 1-kFA that accepts L_m.*

Proof. Assume that there is a 1-kFA that accepts L_m. In particular, it must accept all the words in the set

$$L_m^\nu = \{w_1 * \ldots * w_{2m} \mid w_i \in \{0,1\}^\nu, w_i = w_{2m+1-i}, 1 \leq i \leq 2m\}.$$

That is, the sublanguage of L_m where all the subwords have a fixed length ν. We now define the *type* of a configuration $(q, j_1, \ldots, j_k)$ as the tuple $(\lceil j_1/(\nu+1) \rceil, \ldots, \lceil j_k/(\nu+1) \rceil)$. The type of a configuration specifies which word (or following delimiter) is being read by each of the heads.

Let $c_1(x), c_2(x), \ldots, c_{\ell_x}(x)$ be a successful run of $\mathcal{A}$ over a word $x \in L_m^\nu$ (where ℓ_x is the length of this run). Let now $d_1(x), \ldots, d_{\ell'_x}(x)$ be the subsequence obtained by selecting $c_1(x)$ and all subsequent $c_i(x)$ such that $type(c_i(x)) \neq type(c_{i-1}(x))$ (that is, the subsequence of configurations where at least one head changes the word it is looking at). We call this subsequence a *pattern* of x. Since $\ell'_x \leq k \cdot (2m-1)+1$, there can be at most

$$P = (|Q| \cdot (2m(\nu+1))^k)^{k \cdot (2m-1)+1}$$

patterns.

We now partition the set L_m^ν according to their patterns. As $|L_m^\nu| = 2^{m\nu}$, there must be a set S_0 with at least $2^{m\nu}/P$ words. Let $\hat{d}_1, \ldots, \hat{d}_{\hat{\ell}}$ be the pattern that defines S_0.

Claim. *For any run of $\mathcal{A}$ on an $x \in L_m^\nu$, there exist i such that w_i* and $w_{2m+1-i}*$ (or $w_{2m}\lhd$ if $i=1$) are never being read simultaneously.*

Proof of Claim. It two heads read such a matched pair of words simultaneously, then at no other point in the run could the same pair of heads read another matched pair of words. Since there are only $\binom{k}{2}$ pairs of heads but $m > \binom{k}{2}$ matched pairs of words, the claim follows. □

Notice that the i given in this claim depends exclusively on the pattern of the run. Let i_0 be such a value for the pattern $\hat{d}_1, \ldots, \hat{d}_{\hat{\ell}}$. We can then partition S_0 into classes according to the string

$$w_1 * w_2 * \cdots * w_{i_0-1} * w_{i_0+1} * \cdots * w_{2m-i_0} * w_{2m+2-i_0} * \cdots * w_{2m}.$$

That is, two words in S_0 belong to the same class iff they are identical except in w_{i_0}, w_{2m+1-i_0}.

Let now S_1 be a class that contains at least

$$|S_0|/2^{\nu(m-1)} \geq 2^\nu/P$$

words, and let ν be large enough so that $|S_1| \geq 2$.

Let now $x = x_1 * \cdots * x_{2m}$ and $y = y_1 * \cdots * y_{2m}$ be two distinct words in S_1. By assumption $x_j = y_j$ iff $j \notin \{i_0, 2m+1-i_0\}$. We claim that the word $z = x_1 * \cdots * x_{2m-i_0} * y_{2m+1-i_0} * \cdots$ obtained by replacing x_{2m+1-i_0} in x with y_{2m+1-i_0} is accepted by $\mathcal{A}$, but $z \notin L_m$ since $z_{i_0} \neq z_{2m+1-i_0}$, which gives us the desired contradiction.

Let $c_1(x), \ldots$ and $c_1(y), \ldots$ be successful runs of x and y, respectively. By construction, both runs contain the pattern $\hat{d}_1, \ldots, \hat{d}_{\hat{\ell}}$ as a subsequence. We divide the runs into $\hat{\ell}$ blocks each by beginning a new block with each occurrence of an element $\hat{d}_j$.

By definition of the pattern, the subwords of x or y being read change only at the transitions between blocks; during any given block, they remain fixed. Since the runs have the same pattern, during the i-th block the heads are reading the corresponding subwords of x and y.

We construct an accepting run for $\mathcal{A}$ on z by selecting successive blocks from the run for x $\{c_i(x)\}$, except when $\mathcal{A}$ would be reading during that block $x_{2m+1-i_0} \neq z_{2m+1-i_0}$, in which case, we select the corresponding block from the run for y $\{c_i(y)\}$ (since $y_{2m+1-i_0} = z_{2m+1-i_0}$). This sequence is a valid computation for z since the last configuration in block i for either run yields $\hat{d}_{i+1}$ as next configuration of $\mathcal{A}$, and by construction $\mathcal{A}$ never reads i_0 and $2m+1-i_0$ simultaneously, so, at any moment, $\mathcal{A}$ behaves exactly as it would if the input had been x or y. □

Corollary 3.5. *For all $k \geq 1$, $k+1$ head (deterministic) multihead automata are more expressive than k head (deterministic) multihead automata.*

Proof. Let $m = \binom{k+1}{2}$. The language L_m is accepted by a 1-k+1DFA (Lemma 3.3) (in particular by a 1-$k+1$FA) but not by any 1-kFA (Lemma 3.4) (in particular by no 1-kDFA). □

However, two heads suffice for accepting the complement of L_m.

Lemma 3.6. *For all $m \in \mathbb{N}$, there is a 1-2FA that accepts $\overline{L_m}$.*

Proof Sketch. The automaton non-deterministically guesses if the input word does not match the input syntax, or if there are two subwords w_i, w_{2m+1-i} that are not equal. It then needs at most two heads to verify its guess. □

This, in particular, means that non-deterministic k-head automata (with a fixed k) are not closed under complement. Furthermore, since 1-kDFA are closed under complement (Exercise!), the following corollary follows.

Corollary 3.7. *For every $k \geq 1$ there is a language that is recognized by a 1-2FA but by no 1-kDFA.*

This, however, says nothing about the limit. That is, we still do not know whether the family of all non-deterministic one-way multihead automata is strictly more expressive than that of deterministic one-way multihead automata. To see that this is the case, we consider other related languages:

$$M_m = \{w_1 * w_2 * \ldots * w_{2m} \mid w_i \in \{0,1\}^*, \exists i. w_i \neq w_{2m+1-i}\},$$

$$M = \bigcup_{m \in \mathbb{N}} M_m.$$

Notice that M is not $\overline{L_m}$.

Lemma 3.8. *There is a 1-3FA that accepts M.*

Proof. To recognize this language, we can send heads one and two to the beginning of some (non-deterministically chosen) subword w_i. We use head one to count the number of words between w_i and the end marker $\triangleleft$, which is used to (simultaneously) position head three at the beginning of word w_{2m+1-i}. At this point, we have head two pointing at the beginning of w_i and head three at word w_{2m+1-i}. Thus, we can use them both to verify that $w_i \neq w_{2m+1-i}$. □

Lemma 3.9. *There is no 1-kDFA that accepts M*

Proof. Suppose there is a 1-kDFA that accepts M. Then, there is also a 1-kDFA that accepts $\overline{M}$. This implies that, for any fixed m, the language

$$L_m = \overline{M} \cap \{w_1 * \ldots * w_{2m} \mid w_i \in \{0,1\}^*, 1 \leq i \leq 2m\}$$

is accepted by a 1-$k+1$DFA: one needs only an additional head to count up to $2m$.

Hence, if we choose $m > \binom{k+1}{2}$ we obtain a contradiction to Lemma 3.4. □

Corollary 3.10. *Nondeterministic one-way three-head automata are strictly more expressive than deterministic one-way multihead automata.*

Note. It is in fact possible to improve the result from Corollary 3.10: nondeterministic one-way two-head automata are strictly more expressive than deterministic one-way multihead automata (Exercise?).

We now show that one-way deterministic multihead automata are not closed under concatenation, union and intersection, with the help of the following languages:

$$\begin{aligned}
C_m &= \{u * w \mid u \in \{0,1\}^*, w \in L_m\}, \\
D_m &= \{w\#w \mid w \in C_m\}, \\
E_m &= \{u * w \mid u \in \{0,1\}^*, w \in M_m\}, \\
F_m &= \{u * w\#v * w \mid u \neq v \in \{0,1\}^*, w \in L_m\}.
\end{aligned}$$

Lemma 3.11. *Let $k \in \mathbb{N}$ and $m = \binom{k}{2}$. For any language L such that*

1. *$C_m \cup D_m \subseteq L$, and*
2. *$L \cap (E_m \cup F_m) = \emptyset$,*

there is no 1-kDFA that accepts L.

Proof Sketch. The proof follows the same basic idea of Lemma 3.4. Suppose that there is a 1-kDFA $\mathcal{A}$ that accepts L and let $y \in C_m \cup D_m$. We can look at the patterns of an accepting run of y, up to the point in which one head has read the whole subword word $y_1 = u * w$ of y (that is, the part of the run that would be equivalent to the words y_1 and $y_1\#y_1$, since $\mathcal{A}$ is deterministic).

There are two possible cases: either (i) every head has already read the initial subword $u \in \{0,1\}^*$, or (ii) there is at least one head that has not read that initial subword.

For the case (i), we consider words $y_1\#y_1 \in D_m$, where y_1 has the pattern described before. Since at this point all heads have read the subword u in the part before the symbol $\#$, but none has read any symbol after $\#$, this automaton will accept also a word $u * w\#v * w$, where $u \neq w$. But by assumption 2. this word is not in L.

For the case (ii) we consider the words in C_m. Then at this point we have a head that is at the end of the word, while one head has not yet read the initial subword u. Hence, there is an $i_0, 1 \leq i_0 \leq m$ such that no pair of heads is reading both instances of the word w_{i_0} at the same time. As in the proof of Lemma 3.4, we can produce a word in E_m that is accepted by $\mathcal{A}$, which yields a contradiction. □

Theorem 3.12. *One-way k-head automata are not closed under concatenation, union, or intersection.*

Proof. Consider the languages

$$\begin{aligned} G_1 &= \{0,1\}^* * \cup\{\varepsilon\}, \\ G_2 &= \{w\#w \mid w \in \{0,1,*\}^*\} \cup \{\varepsilon\}. \end{aligned}$$

It is easy to see that G_1 is accepted by a 1-1DFA, and G_2 by a 1-2DFA. Let now $m = \binom{k}{2}$.

The language $G_1 \cdot G_2 \cdot L_m$ is not accepted by any 1-kDFA, because it satisfies the conditions of Lemma 3.11:

1. $C_m \subseteq G_1 \cdot \{\varepsilon\} \cdot L_m \subseteq G_1 \cdot G_2 \cdot L_m$, and $D_m \subseteq \{\varepsilon\} \cdot G_2 \cdot \{\varepsilon\} \subseteq G_1 \cdot G_2 \cdot L_m$.
2. Since E_m has no symbol $\#$, we get that $G_1 \cdot G_2 \cdot L_m \cap E_m = G_1 \cdot \{\varepsilon\} \cdot L_m \cap E_m = \emptyset$, because in E_m there is a j such that $w_j \neq w_{2m+1-j}$. Since the words in F_m have only one symbol $\#$ in the middle, it holds that $G_1 \cdot G_2 \cdot L_m \cap F_m = \{\varepsilon\} \cdot G_2 \cdot \{\varepsilon\} = \emptyset$.

Thus, 1-kDFA are not closed under concatenation.

For the closure under union, notice that $\{0,1\}^* * \cdot L_m$ is accepted by a 1-kDFA, but the language $G_2 \cup \{0,1\}^* * \cdot L_m$ satisfies the conditions of Lemma 3.11 (Exercise!).

Since 1-kDFA are closed under complement, this also shows that they are not closed under intersection. □

Using similar arguments, it is possible to prove that 1-kDFA are not closed under Kleene star, reversal, and other operations.

For non-deterministic multihead automata things are slightly different. In fact, we know already that, contrary to the deterministic case, non-deterministic k-head automata are not closed under complement. It is also easy to see that this class of automata is closed under union: given two 1-kFA $\mathcal{A}_1, \mathcal{A}_2$ that accept languages L_1 and L_2, respectively, one simply constructs the automaton $\mathcal{A}$ as the disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}$ accepts the language $L_1 \cup L_2$. However, this class is not closed under intersection. One can construct two languages that are accepted by 1-kFA, but whose intersection yields the language L_{m+1} with $m = \binom{k}{2}$ (Exercise!). As shown by these three specific closure properties, deterministic and non-deterministic multihead automata may have very different behaviours with respect to distinct operators.

For two-way automata we have the following properties. (1) two-way deterministic/non-deterministic k-head automata are closed under intersection: for recognizing $L_1 \cap L_2$, simply run the automaton that accepts L_1, but upon reaching a final state, "restart" the automaton by setting all the heads back to the initial symbol $\rhd$, and then run the automaton accepting L_2; (2) two-way non-deterministic k-head automata are closed under union. All other closure properties for this family of automata are left open in this course.

Perhaps more interesting is the fact that the emptiness problem for one-way three-head deterministic automata is undecidable. That is, by allowing two more heads in the automaton we jump from a polynomial time emptiness test (in DFA) to undecidability.

Theorem 3.13. *The emptiness problem for 1-3DFA is undecidable.*

Proof. We show that we can build a deterministic automaton that accepts the halting computations of deterministic Turing machines on a blank tape. Since it is undecidable whether such computations exist in general, we obtain the desired result.

Let Z be a deterministic TM, and $w_Z = v_1 * v_2 * \ldots * v_n$ be the halting computation of Z on a blank input tape. More clearly, v_1 is the initial configuration, v_n is the halting configuration and v_{i+1} is the direct successor of v_i. Assume w.l.o.g. that $n \geq 2$

The automaton works as follows: head 1 reads the state and symbol of the configuration, while heads 2 and 3 check that each configuration is compatible with its predecessor given the instructions of the TM. The following example shows the important details of this construction [handwritten example].

Heads 2 and 3 move simultaneously over consecutive configurations, checking that the symbols in the tape are the same (except for that written by the TM-head in the last action) and that the TM-head is in the correct position, according to the instructed movement. Afterwards, head 1 moves to the symbol representing the internal state of the TM in the following configuration, confirms that it is correct, and reads the associated symbol to check on the next instruction. If the instruction is to HALT, then head 3 must be reading the symbol $\lhd$ (and only then).

This automaton accepts the word w_Z (and only this string) iff Z halts. This shows that the emptiness problem for 1-3DFA is undecidable. □

Note. It was recently shown (in 2008) that, if one allows two-way movements, then two heads are enough; i.e., the emptiness problem for 2-2DFA is undecidable. It is still an open problem whether the emptiness problem 1-2DFA, or even 1-2FA is undecidable too.

4 k-ary Transitive Closure

We now look at a logical representation of the languages accepted by multihead automata. Recall from Section 2.2 that the transitive closure operator uses as its basis a first-order formula with two free variables. Intuitively, this can be thought of as a relation between the (only) head before and after a transition of an automaton. We can then lift this intuition to the multihead case. Now, the formula φ will represent a relation between the k heads before and after a transition. Hence, φ must have $2k$ free variables; i.e., it is of the form $\varphi(x_1, \dots, x_k, y_1, \dots, y_k)$, representing the following binary relation between tuples of k elements: $(x_1, \dots, x_k)\varphi(y_1, \dots, y_k)$ iff $\varphi(x_1, \dots, x_k, y_1, \dots, y_k)$ is true.

As in the case where $k = 1$, we can consider the reflexive, transitive closure of this relation, which we call *k-ary transitive closure* and denote as $\mathsf{TC}_k[\varphi]$. Notice that $\mathsf{TC}_k[\varphi]$ has $2k$ free variables too.

For brevity, when the arity k is known, we will use the $\overrightarrow{x}$ to denote the tuple $x_1, \dots, x_k$.

The semantics of this operator is a simple generalization of the unary transitive closure from Section 2.2: given $2k$ elements $\overrightarrow{d}, \overrightarrow{e}$ of the domain $\mathsf{dom}(I)$, the interpretation I satisfies $\mathsf{TC}_k[\varphi(\overrightarrow{x}, \overrightarrow{y})](\overrightarrow{d}, \overrightarrow{e})$ if $\overrightarrow{d} = \overrightarrow{e}$ or there exist $\overrightarrow{d_1}, \dots, \overrightarrow{d_m} \in \mathsf{dom}(I)^k$ such that the following hold:

- $\overrightarrow{d} = \overrightarrow{d_1}$;
- $\overrightarrow{e} = \overrightarrow{d_m}$; and
- $I \models \varphi(\overrightarrow{d_i}, \overrightarrow{d_{i+1}})$ for all $1 \leq i < m$.

Example 4.1. Let Σ be an alphabet and consider the formula

$$\varphi(x_1, x_2, y_1, y_2) = \bigvee_{a \in \Sigma} Q_a(x_1) \Leftrightarrow Q_a(x_2) \wedge s(x_1, y_1) \wedge s(x_2, y_2)$$

and let

$$\psi = \exists x, y. x > \min \wedge s(y,x) \wedge \mathsf{TC}_2[\varphi](\min, x, y, \max) \wedge \bigvee_{a\in\Sigma} Q_a(y) \Leftrightarrow Q_a(\max).$$

Then $L(\psi) = \{ww \mid w \in \Sigma^*\}$.

We consider also the *deterministic* transitive closure, with the additional restriction that if $\varphi(\overrightarrow{d}, \overrightarrow{e_1})$ and $\varphi(\overrightarrow{d}, \overrightarrow{e_2})$ hold, then $\overrightarrow{e_1} = \overrightarrow{e_2}$. Formally, the k-ary deterministic closure DTC_k is defined in terms of TC_k as follows:

$$\mathsf{DTC}_k[\varphi(\overrightarrow{x}, \overrightarrow{y})] = \mathsf{TC}_k[\varphi(\overrightarrow{x}, \overrightarrow{y}) \wedge \forall \overrightarrow{z}. \varphi(\overrightarrow{x}, \overrightarrow{z}) \Rightarrow \overrightarrow{y} = \overrightarrow{z}].$$

Example 4.2. Let Σ be an alphabet and consider the formula

$$\varphi(x_1, x_2, y_1, y_2) = \bigvee_{a\in\Sigma} Q_a(x_1) \Leftrightarrow Q_a(x_2) \wedge s(x_1, y_1) \wedge s(y_2, x_2)$$

and let

$$\psi = \exists x, y. x > \min \wedge s(x,y) \wedge \mathsf{DTC}_2[\varphi](\min, \max, x, y) \wedge \bigvee_{a\in\Sigma} Q_a(y) \Leftrightarrow Q_a(x).$$

Then $L(\psi) = \{w\overleftarrow{w} \mid w \in \Sigma^*\}$.

Note. The formula ϕ from Example 4.1 actually defines a *deterministic* operator. Hence, changing TC for DTC would yield the exact same language. Recall that 1-2DFA cannot accept the language $\{ww \mid w \in \Sigma^*\}$; however, 2-2DFA can.

We will now show that k-ary transitive closure can encode two-way multihead automata.

Theorem 4.3. *Let $\mathcal{A}$ be a 1-kFA. Then, there exists a first-order formula with k-ary transitive closure ψ such that $L(\mathcal{A}) = L(\psi)$.*

Proof. The main idea is that the variables in a first-order formula with k-ary transitive closure can be seen as pointers to positions in the input word. Thus, using k variables, it is possible to simulate the k heads of the automaton.

Consider the formula

$$\mathsf{mv}(p_{1h}, m, p_{2h}) = \begin{cases} s(p_{1h}, p_{2h}) & \text{if } m = 1 \\ p_{1h} = p_{2h} & \text{if } m = 0 \\ s(p_{2h}, p_{1h}) & \text{if } m = -1. \end{cases}$$

We can then capture a movement of the k heads with the formula

$$\mathsf{mvk}(\overrightarrow{p_1}, \overrightarrow{m}, \overrightarrow{p_2}) = \bigwedge_{l=1}^{k} \mathsf{mv}(p_{1l}, m_l, p_{2l}).$$

This formula is true iff the k heads were at position $\overrightarrow{p_1}$ and have a movement of $\overrightarrow{m}$ cells to the right to reach position $\overrightarrow{p_2}$.

We can then caputure a transition of the automaton by the formula

$$\mathsf{tr}(\overrightarrow{a}, \overrightarrow{m}) = \bigwedge_{l=1}^{k} Q_{a_l}(p_{1l}) \wedge \mathsf{mvk}(\overrightarrow{p_1}, \overrightarrow{m}, \overrightarrow{p_2}),$$

which is true iff such a move was made while the heads were reading the symbols $\overrightarrow{a}$.

We now use the same idea from Theorem 2.8 to encode a finite automaton with a FOL formula. Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a 2-kFA, we construct an equivalent FOL formula with k-ary transitive closure χ. For this, we recursively build formulas $\chi^n_{ij}(\overrightarrow{p_1}, \overrightarrow{p_2})$, which are true iff when $\mathcal{A}$ is started in state i with its heads positioned at $\overrightarrow{p_1}$, then $\mathcal{A}$ can perform 0 or more transitions to reach state j and position its heads at $\overrightarrow{p_2}$, while not passing through states higher than n. Formally, we have

$$\chi^0_{ij}(\overrightarrow{p_1}, \overrightarrow{p_2}) = \bigvee_{\substack{\overrightarrow{a} \in (\Sigma \cup \{\rhd, \lhd\})^k, \\ (i, \overrightarrow{a}, j, \overrightarrow{m}) \in \Delta}} \mathsf{tr}(\overrightarrow{a}, \overrightarrow{m}) \vee \begin{cases} \text{false} & \text{if } i \neq j, \\ \overrightarrow{p_1} = \overrightarrow{p_2} & \text{if } i = j. \end{cases}$$

Note that this formula allows transitions without head movements, via the last disjunction.

For the recursion, we can now set

$$\begin{aligned}\chi^n_{ij}(\overrightarrow{p_1},\overrightarrow{p_2}) =& \exists\overrightarrow{p_3},\overrightarrow{p_4}.(\chi^{n-1}_{in}(\overrightarrow{p_1},\overrightarrow{p_3}) \wedge \mathsf{TC}_k[\chi^{n-1}_{nn}](\overrightarrow{p_3},\overrightarrow{p_4}) \wedge \chi^{n-1}_{nj}(\overrightarrow{p_4},\overrightarrow{p_2}))\\ &\vee \chi^{n-1}_{ij}(\overrightarrow{p_1},\overrightarrow{p_2}).\end{aligned}$$

With these formulas we can now set

$$\chi = \exists\overrightarrow{p_1}. \bigvee_{\substack{q\in I\\ f\in F}} \chi^{|Q|}_{q,f}(\overrightarrow{\min},\overrightarrow{p_1}).$$

This formula states that if $\mathcal{A}$ is started with all the heads pointing at min and in an initial state, then $\mathcal{A}$ can move to some head position $\overrightarrow{p_1}$ in a final state. This is exactly the condition for accepting the input, and hence $L(\mathcal{A}) = L(\chi)$. □

Note. Using a similar argument, it can also be shown that 2-kDFA can be encoded through FOL formulas with deterministic transitive closure. Due to the restrictions of the deterministic operator, several details need to be considered, which makes the proof long, but not more difficult.

Theorem 4.3 shows that every language accepted by a 2-kFA is also accepted by a FOL formula with k-ary transitive closure. It now remains to check whether the converse is also true, that is, whether for every such formula there is an automaton accepting the same language.

In Section 2.2 it was shown that for $k = 1$ the converse does hold. The proof presented there (see Theorem 2.8) depends on a result from Büchi that does not generalize to multihead automata (namely, that MSOL and non-deterministic automata have the same expressive power). This means that we cannot reuse that proof in this more general setting.

To solve this problem, we introduce so-called k-regular formulas. These are syntactic variants of (a subclass of) FOL formulas with k-ary transitive closure, that intuitively generalize regular expressions. We then show that the class of 2-kFA and k-regular formulas have the same expressive power.

Definition 4.4. The set of k-regular formulas over an alphabet Σ is the smallest set such that:

- $\mathsf{tr}(\overrightarrow{a},\overrightarrow{m})$ is a k-regular formula, for all $\overrightarrow{a} \in (\Sigma \cup \{\triangleright,\triangleleft\})^k$ and $\overrightarrow{m} \in \{-1,0,1\}^k$,
- if φ,ψ are k-regular formulas, then so are $\varphi\circ\psi$, $\varphi\vee\psi$, and φ^*.

The semantics of these expressions are given through a translation to FOL+TC_k. We define the translation T from k-regular formulas to FOL+TC_k inductively as follows:

- $T(\mathsf{tr}(\overrightarrow{a},\overrightarrow{m})) = \bigwedge_{l=1}^k (Q_{a_l}(p_{1l}) \wedge \mathsf{mv}(p_{1l},m_l,p_{2l}))$ (cf. Theorem 4.3),
- $T(\varphi\circ\psi) = \exists\overrightarrow{p_3}.(T(\varphi)_{[\overrightarrow{p_2}/\overrightarrow{p_3}]} \wedge T(\psi)_{[\overrightarrow{p_1}/\overrightarrow{p_3}]})$, where $\theta_{[\overrightarrow{x}/\overrightarrow{y}]}$ denotes the substitution of all free occurrences of variables from $\overrightarrow{x}$ with the respective constants or variables from $\overrightarrow{y}$,
- $T(\varphi\vee\psi) = T(\varphi)\vee T(\psi)$,
- $T(\varphi^*) = \mathsf{TC}_k[\varphi](\overrightarrow{p_1},\overrightarrow{p_2})$.

Notice that the free variables in all these formulas are $\overrightarrow{p_1},\overrightarrow{p_2}$. The language accepted by a k-regular formula φ is given by:

$$L(\varphi) = L(\exists\overrightarrow{p_2}.T(\varphi)_{[\overrightarrow{p_1}/\overrightarrow{\min}]}).$$

An important thing to consider is that, despite its syntactical similarity, it is not the case that $L(\varphi\circ\psi) = L(\varphi)\circ L(\psi)$. Likewise, $L(\varphi^*) \neq L(\varphi)^*$.

Using a construction similar to the proof of Theorem 4.3, it is possible to show that for every 2-kFA $\mathcal{A}$ there is a k-regular formula φ with $L(\mathcal{A}) = L(\varphi)$ (Exercise!).

Theorem 4.5. *For every k-regular formula φ there is a 2-kFA $\mathcal{A}$ such that $L(\varphi) = L(\mathcal{A})$.*

Proof. The proof follows the same idea used for building a non-deterministic automaton that accepts a regular expression. For each subformula of φ, we inductively define an automaton $\mathcal{A}$ that accepts the same language, with the following properties:

- $\mathcal{A}$ has exactly one initial and one final state,

- there is no transition leading to the initial state, and
- there is no transition from the final state.

For the base case, that is for a formula of the form $\mathsf{tr}(\overrightarrow{a}, \overrightarrow{m})$, we have an automaton having only two states and one transition [handwritten image a)]. Clearly, this automaton satisfies the three conditions above and accepts the desired formula.

We now look at the inductive step. Assume that we have automata $\mathcal{A}_1$ and $\mathcal{A}_2$ accepting φ and ψ, respectively, satisfying the three conditions above. For each of the constructors we give an automaton that accepts the corresponding formula:

- $\varphi \vee \psi$: for this case, we want to choose which of the automata $\mathcal{A}_1$ or $\mathcal{A}_2$ is used for accepting the input word. Thus, we can actually set these automata independently, and get two initial and final states. This, however, would violate the first condition of our automata. Thus, what we do is to introduce two new states that will act as the new initial and final states of the automaton, respectively, and connect them via transitions that do not change the positions of the heads [image b)].

  It is easy to see that this automaton satisfies the three conditions, and that it accepts the language defined by $\varphi \vee \psi$.

- $\varphi \circ \psi$: in this case we want to combine the two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ sequentially. This is easily done by including a new motionless transition from the final state of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$ [image c)].

- φ^*: this case is very similar to the previous one, but the sequential use of automaton $\mathcal{A}_1$ must be able to repeat as many times as one wants (even 0). Hence, we need again to introduce new states that will now act as initial and final state, and add motionless transitions to the automaton $\mathcal{A}_1$ [image d)].

  Once again, it is easy to see that this automaton satisfies the three conditions above and that it accepts the desired language φ^*.

□

Notice that k-regular formulas are intrinsically non-deterministic. Indeed, not only the disjunction constructor, but also the star (given by the transitive closure) yield a non-deterministic automaton. Although it is potentially possible to redefine the * operator to use the deterministic transitive closure, this would not suffice for having a (directly) deterministic automata construction. It is thus unclear which kinds of restrictions one could impose to obtain a subclass of $\mathsf{FOL{+}DTC}_k$ that expresses exactly the languages accepted by deterministic multihead automata.

This means that we have a precise logical characterization of the languages accepted by non-detereministic two-way multihead automata, but only a (possibly) over-characterization for the deterministic case.

5 Tree Walking Automata

We will now consider a special kind of automata that are able to recognize tree languages. For simplicity, we will consider only *binary* trees. It should be noted, however, that the ideas presented in this chapter can be easily extended to any arbitrary (finite) arity.

Definition 5.1. Let Σ be an alphabet. A Σ-*tree* t is a mapping from $\mathsf{N}_t \subseteq \{1,2\}^*$ to Σ, where N_t is a finite, non-empty, prefix-closed set such that for any $v \in \mathsf{N}_t$ it holds that $v1 \in \mathsf{N}_t$ iff $v2 \in \mathsf{N}_t$. N_t is the set of *nodes* of t.

Intuitively the elements in N_t represent the positions in the tree. ε represents the root node and the nodes $v1$ and $v2$ are the left- and right-successor of v, respectively. We force trees to be binary in the sense that every node has either none or exactly two successors.

We assume that for every node we know whether it is the root node, or a leaf node, and also which number of child it is. This is called the *type* of the node. Formally, the type of a node is an element of $\mathsf{tp} = \{r,1,2\} \times \{l,i\}$, where r stands for the root, 1 for a left son, 2 for a right son, l for a leaf, and i for an internal (non-leaf) node [leaf nodes are those that have no successors; i.e., $v \in \mathsf{N}_t$ is a leaf node iff $v1 \notin \mathsf{N}_t$].

We will introduce the notion of a *tree-walking automaton.* Basically, this automata are a straightforward generalization of two-way automata capable of dealing with trees as inputs. Hence, a head of the automaton can still move backwards (to the unique parent of the corresponding node, if it is not the root), stay at the same node, or move forward. However, in this last case, there are two possible options: either move to the left- or the right- successor. We will represent this by a movement of 1 or 2, respectively.

Definition 5.2. A *k-head tree-walking automaton* is a tuple of the form $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, where Q, Σ, I and F are as in multihead automata (over words) and

$$\Delta \subseteq Q \times \mathsf{tp}^k \times \Sigma^k \times Q \times \{-1,0,1,2\}^k$$

is the transition relation.

As seen by this Δ, the transitions of the automaton do not only depend on the symbols being read, but also on the type of the nodes at which the heads of the automaton are looking. This enables us to distinguish whether the automaton is on the first or second branch of a subtree, and removes the need of having initial and final markers ($\rhd, \lhd$) used in multihead word automata.

The notion of deterministic automaton requires also to take care of the types. Formally, the automaton $\mathcal{A}$ is deterministic if for every $(q, \overrightarrow{\mathsf{t}}, \overrightarrow{a}) \in Q \times \mathsf{tp}^k \times \Sigma^k$ there is at most one transition of the form $(q, \overrightarrow{\mathsf{t}}, \overrightarrow{a}, q', \overrightarrow{m}) \in \Delta$.

As for the word case, a *configuration* of $\mathcal{A}$ on a tree t over Σ is a pair $(q, \overrightarrow{u})$ where $q \in Q$ and $\overrightarrow{u}$ is a k-tuple of nodes of t indicating the positions of the k-heads. A *run* on t is a sequence of configurations where every two consecutive configurations are consistent with the transition relation Δ. Such a run is *accepting* if it starts with a configuration $(q_0, \overrightarrow{\varepsilon})$ with $q_0 \in I$ and finishes with a configuration $(q_m, \overrightarrow{u})$ with $q_m \in F$.

Example 5.3. Let $\Sigma = \{a, b\}$, and consider the language L of all trees where no leaf node is labeled with a.

The language L is accepted by the following one-head tree-walking automaton [hand-written page]. Notice that the automaton is even deterministic. The idea is that it traverses systematically the whole input tree, to ensure that no leaf is left unchecked, but since there are no transitions available if the leaf being read is labeled with a, this traversal fails whenever t is not in L. As a particular example, consider the accepting run on the input tree [a) hand-written]

There exist several characterizations of *regular* tree languages. Here we use the one based on FOL with transitive closure: a tree langue L is *regular* iff there is a FOL+DTC formula φ *with two successor functions* such that $L(\varphi) = L$.

Notice that we can also use the abbreviations $\mathsf{leaf}(x), \mathsf{root}(x)$ that represent that x is a leaf or a root node, respectively. Using the same transformation from (word) automata to FOL+DTC from Section 2.2, but now considering both successor functions to distinguish between the first and second child of a node, one can prove that every tree language accepted by a one-head tree walking automaton is regular. However, the converse is not true. That is, there are regular languages that are not accepted by any one-head TWA.

Example 5.4. Let $\Sigma = \{a, b\}$. We call an internal node v in a tree t *branching* if the left and right subtrees of v both contain a leaf labeled with a. Consider the language L consisting of all trees where no internal node is labeled with a and such that the path to each a-labeled leaf contains an even number of branching nodes. [see handwritten example]

The language is regular. To show this, we will construct a FOL+DTC formula that accepts it. Consider the formula $\mathsf{branch}(y)$ that specifies that y is a branching node:

$$\begin{aligned}\mathsf{branch}(y) =& \exists z_1, z_2. \mathsf{leaf}(z_1) \land \mathsf{leaf}(z_2) \land z_1 >_1 y \land z_2 >_2 y \land \\ & Q_a(z_1) \land Q_a(z_2).\end{aligned}$$

We can then define the functions $\psi_1(x, y)$ and $\psi_2(x, y)$ that express that y is the lowest, and second lowest branching ancestor of x, respectively:

$$\begin{aligned}\psi_1(x, y) =& y < x \land \mathsf{branch}(y) \land \\ & \forall z. y < z < x \Rightarrow \neg \mathsf{branch}(z), \\ \psi_2(x, y) =& \exists z. \psi_1(x, z) \land \psi_1(z, y).\end{aligned}$$

Notice that both ψ_1 and ψ_2 are functional. We can now define the formula

$$\varphi = \forall x. Q_a(x) \Rightarrow \mathsf{leaf}(x) \land \exists y. \mathsf{DTC}[\psi_2](x, y) \land \neg \exists z. \psi_1(y, z)$$

that accepts the language L.

However, this language cannot be accepted by any 1-TWA (the proof is too technical and complex and will not be presented in this lecture).

Once we allow the use of multiple heads, then we get again out of the realm of regular languages. Let L be a word language and consider the tree language

$$L_t = \{t \mid \text{the left-most branch of } t \text{ is labeled with a word } w \in L\}.$$

It is easy to see that L is accepted by a 2-kFA iff L_t is accepted by a k-TWA without transitions to the right-child (that is, using only head movements in $\{-1, 0, 1\}$). In fact, the same automaton can be used to accept both languages. Thus, for instance, for the language $L = \{ww \mid w \in \Sigma^*\}$, L_t is recognized by a 2-TWA but clearly L_t is not regular.

The relation between k-head TWA and k-ary transitive closure translates directly from the word case to the tree case: every language expressible through a (deterministic) k-TWA is also expressible through FOL with k-ary (deterministic) TC. However, the converse is not clear. Again, for non-deterministic automata and operators, one could define a notion of k-regular formulas that exactly characterize the languages recognized by k-TWA. However, the deterministic case would still be left unclear.

We will now follow the opposite approach. Rather than restricting the class of formulas that can be used, we further generalize the notion of k-head automata. This generalization allows us to mark specific nodes of the tree that have been visited by the automaton, and access and modify this marks during the execution of the automaton. Thus, one can have more than just the local information of what the automaton is currently looking at. This markings are kept in the form of so-called "pebbles" that the automaton can drop and retrieve in a LIFO fashion. We will see that automata with this kind of pebbles fully characterize FOL+TC.

Definition 5.5. A *k-head tree-walking pebble automaton* is a tuple of the form $\mathcal{A} = (Q, \Sigma, I, \Delta, F, X)$ where Q, Σ, I and F are as usual and X is a finite set of *pebbles*. To define the transition relation, we first introduce the sets of *tests* and *consequences*:

$$\begin{aligned}\mathit{tests} =& (\mathsf{tp}^k \times \Sigma^k) \cup (X \times \{i, -i \mid 1 \leq i \leq k\}), \\ \mathit{conqs} =& \{-1, 0, 1, 2\}^k \cup \{\mathit{rtrv}\} \cup \{d_i \mid 1 \leq i \leq k\}.\end{aligned}$$

Then,

$$\Delta \subseteq Q \times \mathit{tests} \times Q \times \mathit{conqs}$$

is the *transition relation*.

Intuitively, there are two kinds of situations that can be verified to decide the applicability of a transition: we can either test that the k heads are reading a sequence of symbols from Σ (as in regular automata), or test whether the i-th head is positioned (or not) at the location of a pebble $x \in X$. These different possibilities are expressed through the set *tests*.

As for the outcome of a transition application, we have a similar dichotomy: one can decide to change the positions of the heads, or perform some pebble-related action. In the latter case, we can either retrieve the last pebble (expressed through *rtrv*) or to drop a new pebble in the current position of head i (d_i).

The semantics of these automata is also defined by means of configurations. In this case, a *configuration* of an automaton $\mathcal{A}$ on a tree t over Σ is a triple $(q, \overrightarrow{u}, \alpha)$, where $q \in Q$ is a state, $\overrightarrow{u}$ is a tuple of k nodes of t indicating the positions of the heads and α is a sequence of pebbles dropped at their positions: $\alpha = (x_1, w_1) \dots (x_m, w_m)$ where $x_i \in X, w_i \in \mathsf{N}_t$.

Two configurations $(q, \overrightarrow{u}, \alpha), (q', \overrightarrow{v}, \beta)$ are *compatible* with $\mathcal{A}, t$ if there is a transition $(q, tst, q', cq) \in Q \times tests \times Q \times conqs$ such that

$$\begin{cases} (type(\overrightarrow{u}), t(\overrightarrow{u})) = tst & \text{if } tst \in \mathsf{tp}^k \times \Sigma^k \\ (x, u_i) \in \alpha & \text{if } tst = (x, i) \\ (x, u_i) \notin \alpha & \text{if } tst = (x, -i), \end{cases}$$

and

$$\begin{cases} \overrightarrow{v} = \overrightarrow{u} + cq, \alpha = \beta & \text{if } cq \in \{-1, 0, 1, 2\}^k \\ \overrightarrow{v} = \overrightarrow{u}, \alpha = \beta(x_m, w_m) & \text{if } cq = rtrv, \alpha = (x_1, w_1) \dots (x_m, w_m) \\ \overrightarrow{v} = \overrightarrow{u}, \beta = \alpha(x_{m+1}, u_i) & \text{if } cq = d_i \text{ and } |\alpha| = m. \end{cases}$$

A run un t is a sequence of configurations where every two consecutive configurations are compatible with $\mathcal{A}, t$. It is called *accepting* if it starts with a configuration $(q_0, \overrightarrow{\varepsilon}, \varepsilon)$ with $q_0 \in I$ and finishes with a configuration $(q_n, \overrightarrow{u}, \alpha)$ with $q_m \in F$.

Notice that this semantics guarantees that the pebbles are used in a nested (LIFO) manner: the command "retrieve" can only pick up the last pebble that was set down, and the command for setting a new

pebble cannot express which specific element of X is used, but it must be the next one available.

Example 5.6. Consider the language over $\Sigma = \{a, b\}$ that consists of all trees having at least one branching node labeled with a. We show that we can easily accept this language with a non-deterministic 1-TWA with one pebble. The automaton non-deterministically choses an internal node labeled with a, marks it with a pebble, and then (non-deterministically) checks that both its right- and left-subtrees contain a leaf labeled with a. [hand-written example]

Notice that this same language can also be accepted by a deterministic 1-TWA with one pebble. The main difference is that the automaton must test each node systematically to decide whether it is branching or not. For that purpose, the automaton must repeatedly drop and retrieve the pebble at (possibly all) the internal nodes labeled with a.

The language L from Example 5.4 is accepted by a deterministic 1-TWA with two pebbles. The idea is to systematically place the first pebble on each a-labelled leaf. For each such position, we follow the path upwards to the root counting the number of branching nodes. To test whether a node is branching, we only need to drop the second pebble at the node and check that its other subtree contains a leaf labeled with a. The pebble is used to recognize that we have returned to the desired node. We then pick up this second pebble and use it for testing the predecessor node. Once the root is reached, we reject if there is an odd number of branching nodes in a path, otherwise, we return to the position of the first pebble and continue. (Exercise!)

We now show that adding pebbles suffices for representing all FOL formulas with transitive closure. We first show this result for the deterministic case, which was the most problematic one. Then we explain the changes that must be made for the non-deterministic case.

Theorem 5.7. *Let $k \geq 1$. For every FOL+DTC_k formula φ there is a deterministic k-TWA with pebbles $\mathcal{A}$ such that $L(\varphi) = L(\mathcal{A})$.*

Proof. The proof is by induction on the structure of the formula. For each FOL+DTC_k formula we construct a deterministic TWA with nested pebbles that always halts (with all its heads at the root). This requirement will be specially useful when dealing with negation, but also for the disjunction and existential quantification cases. Informally, each variable of the formula will can be seen as a pebble for the automaton. For the k-ary transitive closure we will need $3k$ pebbles for testing the formula ($2k$ for marking the initial and final tuples, and k for testing the transitive closure).

Notice that subformulas of φ may contain free variables; hence, we extend the notion of *recognizing* a tree with an automaton: a valuation of the free variables is fixed by putting pebbles on the tree (one per variable), and the automaton should evaluate the formula according to this assignment. Formally, let $\phi(x_1, \ldots, x_n)$ be a formula with free variables $x_1, \ldots, x_n$. The automaton $\mathcal{A}$ for ϕ should check whether $t \models \phi(u_1, \ldots, u_n)$ for nodes $u_1, \ldots, u_n$ in a tree t as follows: it is started in the initial state with all heads at the root of the tree t, where $u_1, \ldots, u_n$ are marked with pebbles $x_1, \ldots, x_n$, respectively. During the computation, $\mathcal{A}$ may use additional pebbles (in the same LIFO way) and it may test $x_1, \ldots, x_n$, but it is not allowed to retrieve these. The computation should halt again with all heads at the root with the original configuration of pebbles. The halting state is accepting iff $t \models \phi(u_1, \ldots, u_n)$.

The base case, for the atomic formulas, is a straightforward construction of single head automata. For instance, consider the formula $s_1(x, y)$. The automaton searches the node that has pebble x and then decides whether it has a child (or is a leaf), and moves to the first child, where it checks that the pebble y is present. Equality, root, leaf, and other atomic formulas can be checked similarly.

Let now $\phi = \neg\phi_1$. Then, we can take the original (deterministic) automaton for ϕ_1 and change the final states to the complementary set. Notice that this only works due to our assumption that the automata being built are always halting. Conjunction and disjunction are also simple. Since we assume that our automata always halt at the root node, we can run the two automata for the subformulae sequentially.

For the quantification $\forall x.\phi_1$, the automaton systematically traverses through the tree (with one head) and places the pebble x at each position, returning then to the root and running the automaton for ϕ_1. If this accepts, the automaton then searches the pebble x and moves it to the next position, effectively testing all possible positions. The existential quantification is treated similarly.

The only remaining constructor is the $\mathsf{DTC}_k[\phi_1](\overrightarrow{x}, \overrightarrow{y})$ with ϕ_1 functional. Here we can use the automaton that accepts ϕ_1 repeatedly to move through the transitive closure. However, we cannot do this directly, as we can end up in a loop, which will contradict the assumption that the automaton always halts. Notice that if, instead of testing this directly we do it backwards, then any such loop must include $\overrightarrow{y}$. Hence, we can easily verify whether a loop exists or not. This is the only case where we will need the k heads of the automaton.

Consider the formula $\phi = \mathsf{DTC}_k[\phi_1](\overrightarrow{x}, \overrightarrow{y})$. Given a tree that has $2k$ nodes marked by pebbles $\overrightarrow{x}, \overrightarrow{y}$, we have to construct an automaton that decides whether we can connect $\overrightarrow{x}$ and $\overrightarrow{y}$ by a series of intermediate tuples satisfying ϕ_1. We use then $3k$ pebbles $\overrightarrow{x'}, \overrightarrow{y'}, \overrightarrow{z'}$ in the following way. The pebbles $\overrightarrow{y'}$ are set first at the position of $\overrightarrow{y}$, and $\overrightarrow{x'}$ are set systematically in all possible positions to check whether $\phi_1(\overrightarrow{x'}, \overrightarrow{y'})$ holds. When we find an adequate tuple $\overrightarrow{x'}$ we move all the pebbles from $\overrightarrow{y'}$ to that position and repeat. If no such tuple exist, then we have seen the wrong path of ϕ_1 predecessors, and hence we need to find the next possible path – unless $\overrightarrow{y'}$ equals $\overrightarrow{y}$. To do this, we set $\overrightarrow{z'}$ at the (unique) positions where $\phi_1(\overrightarrow{y'}, \overrightarrow{z'})$ holds, and then traverse all positions to find the next tuple such that $\phi_1(\overrightarrow{x'}, \overrightarrow{z'})$ is satisfied (for this we need some kind of total order between tuples of nodes; for instance a lexicographical order can be used). During all this tests we have to take care of never setting $\overrightarrow{x'}$ in the same positions as $\overrightarrow{y}$ so as to avoid reaching a loop. This process will eventually find out that $\overrightarrow{x'}$ is at the same position as $\overrightarrow{x}$ (in which case, accepts) or fails to find a new tuple of nodes that satisfy the inverse relation of ϕ_1 (and rejects). $\square$

Notice that, although the number of heads is fixed by the arity of the transitive formulas, the number of pebbles depends on the structure of the formula: we have 1 pebble for each nested quantifier, and $3k$ for each nested transitive closure.

If we remove the determinism in the transitive closure operator (i.e., we do not require the parameter to be functional), we need to restrict to only *positive* occurrences of the TC_k operator; that is, where the TC_k is in the scope of an even number of negations (this is equivalent to asking that negation is applied to atomic formulas only).

In this case, we can build a non-deterministic k-TWA with pebbles as follows: atomic formulas, their negations, conjunction and universal quantification are treated as above. For disjunction and existential quantification, we can use non-determinism in the obvious way. For $\mathsf{TC}_k[\phi_1](\overrightarrow{x}, \overrightarrow{y})$, we now use the non-determinism to check for a sequence of ϕ_1-connected tuples. Since we are using non-determinism for disjunction and existential quantification, and negation appears only in atomic formulas, we do not need to worry about having non-terminating sequences. Non-determinism will make sure that, if the transitive closure holds, the automaton will find it. Thus, we also get the following result.

Theorem 5.8. *Let $k \geq 1$. For every FOL+posTC_k formula φ there is a k-TWA with pebbles $\mathcal{A}$ such that $L(\varphi) = L(\mathcal{A})$.*

We have previously shown that for each automaton we can build a formula that accepts the same language. However, our previous proof did not consider the presence of pebbles. Hence we need to extend our proof to deal with these new elements.

Theorem 5.9. *Let $k \geq 1$. For every k-TWA with pebbles $\mathcal{A}$ there is a FOL+posTC_k formula φ such that $L(\varphi) = L(\mathcal{A})$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F, X)$ be a k-TWA with pebbles. We assume the following w.l.o.g.:

- there are no transitions from final states
- $I \cap F = \emptyset$
- for any transition $(q, tst, q', d_i) \in \Delta$, there is no transition of the form $(q', tst, q'', rtrv)$ (that is, we cannot retrieve a pebble immediately after it has been dropped)
- final states can only be reached when all heads are looking at the root node.

Furthermore, we assume that the states can identify the number of pebbles that have been dropped; that is, if $X = \{x_1, \ldots, x_n\}$, then we can partition the set Q in n sets $Q_0, Q_1, \ldots, Q_n$ where Q_l consists of states where l pebbles are still available, and $I \cup F \subseteq Q_n$. In other words, we can consider that $\mathcal{A}$ consists of $n+1$ "levels" $\mathcal{A}_n, \ldots, \mathcal{A}_0$ such that each $\mathcal{A}_l$ (defined by the set of states Q_l) is a k-TWA with l pebbles $(x_1, \ldots, x_l)$, while the rest of the pebbles have a fixed position in the tree. The intuition is that $\mathcal{A}_l$ works as a TWA that can drop a single pebble x_l and then query $\mathcal{A}_{l-1}$ where to move (and to which state), and then retrieves the pebble x_l again. The formula $\chi^{(l)}_{q,q'}(\overrightarrow{u}, \overrightarrow{v})$ is true if one can perform a computation of $\mathcal{A}_l$ from configuration $(q, \overrightarrow{u}, \alpha)$ to configuration $(q', \overrightarrow{v}, \alpha)$. [Notice the difference with the previous formulas χ]

Consider first the case for which the only consequences of a transition are to move the heads of the automaton (that is, there are no "drop" or "retrieve" instructions). In this case, we are fixed in the same level automaton $\mathcal{A}_l$. Hence, we can simulate any automata instructions with a FOL+DTC_k formula as done in the proof of Theorem 4.3. The only case that has not been covered there is when the test of the transition checks for the i-th head to be where a pebble x_m (or not). This is easily done with the following formulas, respectively:

$$p_{1i} = x_m, \qquad \neg(p_{li} = x_m).$$

Hence, the only instructions we have to deal with are the dropping and retrieval of pebbles.

Since we have assumed that all initial and final states are in Q_n, we have to start with an execution of automaton $\mathcal{A}_n$, and, whenever we drop a pebble in one of these automata, we have to eventually retrieve it. Thus, we can think of this as a step where we drop a pebble, allow the automaton with one less available pebble work, and then retrieve the pebble again.

Thus, the formula $\chi^{(l+1)}_{p,q}(\overrightarrow{u}, \overrightarrow{v})$ is (also) true if $\mathcal{A}_{l+1}$ can drop a pebble to reach state p', $\chi^{(l)}_{p',q'}(\overrightarrow{u}, \overrightarrow{v})$ holds, where the free variable x_l in that formula is replaced by u_i, and $\mathcal{A}_l$ can retrieve a pebble to reach state q. More formally, if

$$\bigvee_{(p,tst_1,p',d_i),(q',tst_2,q,rtrv)} tst_1(\overrightarrow{u}) \wedge tst_2(\overrightarrow{v}) \wedge \chi^{l}_{p',q'}(\overrightarrow{u}, \overrightarrow{v})[x_l/u_i].$$

We can then construct the formula φ given by

$$\varphi = \bigvee_{p\in I, q\in F} \chi^{(n)}_{p,q}(\overrightarrow{\varepsilon}, \overrightarrow{\varepsilon}),$$

which is true if the automaton can reach a final state from an initial state with all the heads pointing at the root. This formula is correct given the assumptions made at the beginning of this proof.

Notice that with this construction, we have only added negations at the atomic formulas (to test that a node is at the place of a given pebble), and hence have no negative use of transitive closure. Thus $\varphi \in pos\mathsf{TC}_k$. □

It is also possible to prove this result for deterministic automata and deterministic transitive closure.

Theorem 5.10. *Let $k \geq 1$. For every deterministic k-TWA with pebbles $\mathcal{A}$ there is a FOL+DTC_k formula φ such that $L(\varphi) = L(\mathcal{A})$.*

Notice that these both last theorems include the case for words, in which we were initially interested. Thus, in particular we know that two-way (deterministic) k-head automata with pebbles and k-ary (deterministic) transitive closure have the same expressive power.

6 Descriptive Complexity

The usual approach to computational complexity is in terms of the time and space necessary to decide a given property $\mathcal{P}$. One can also ask the dual question: how difficult it is to express the property $\mathcal{P}$? As it turns out, these two notions (deciding and expressing) are very closed related, when one considers on the one hand logic, and on the other ordered structures. If we consider the structures and logic we have dealt with in this lecture, we get an interesting characterization.

Let L be the class of all problems solvable in deterministic logarithmic space, and NL its non-deterministic counterpart. We can first characterize these classes by means of two-way multihead automata.

Theorem 6.1.
- $\bigcup_{k=1}^{\infty}$ *k-2DFA*$= \mathsf{L}$,
- $\bigcup_{k=1}^{\infty}$ *k-2FA*$= \mathsf{NL}$.

On the other hand, we obtain a characterization of the same classes by means of first-order logic with transitive closure.

Theorem 6.2.
- $\bigcup_{k=1}^{\infty}$*FOL+*$\mathsf{DTC}_k = \mathsf{L}$,
- $\bigcup_{k=1}^{\infty}$*FOL+*$\mathsf{TC}_k = \mathsf{NL}$.

These two theorems together imply, for instance, that if we do not fix the number of heads of the automata, then the addition of pebbles does not add to the expressivity of multihead automata.

One reason for studying these classes of automata (and logic) is to try to shine light over the $\mathsf{L} = \mathsf{NL}$ problem (similar to the $P = NP$ problem), which is still open.

If, for instance, one could find a language that is accepted by a multihead nondetereministic automaton with pebbles, but not by any multihead deterministic automaton (without pebbles), then we could answer the question negatively. Alternatively, if one can prove that 2-kFA can be determinized (possibly by using more heads), then we could answer the question possitively.

Several open problems in complexity theory can be (and have been) solved through descriptive complexity approaches, the L=NL being only one of them.