1 Introduction

In this lecture, we will learn automata on finite strings and trees (both ranked and unranked) and their relationships with First Order Logic (FO) and Monadic Second Order Logic (MSO).

1.1 Course prerequisites

Basic knowledge of databases, propositional logic, complexity, and theory of computation.

2 Automata on finite strings

A non-deterministic finite state automaton (NFA) \( A \) is a tuple
\[
A = (\Sigma, Q, Q_0, F, \delta),
\]
where

1. \( \Sigma \) is an alphabet,
   Note: \( \Sigma^* \) is a set of (finite) strings over \( \Sigma \).
2. \( Q \) is a set of states,
3. \( Q_0 \) is a set of initial states and \( Q_0 \subseteq Q \),
4. \( F \) is a set of final states and \( F \subseteq Q \),
5. \( \delta : Q \times \Sigma \rightarrow 2^Q \) is a transition function.

A string is a set of positions and labels over some alphabet. Formally, let a set of positions be \( \{0, \ldots, n-1\} \), denoted as \([n-1]\) and a labeling function from positions to members of the alphabet \( \Sigma \) be \( \lambda : \{0, \ldots, n-1\} \rightarrow \Sigma \). For example, if \( S \) is the string \( abaaba \), then the labeling function labels \( \lambda_s(0) = a \), \( \lambda_s(1) = b \), etc. Hence a string \( S \) of length \( n \) is a 2-tuple \( ([n-1], \lambda_s) \).

A run \( \rho_{A,S} : \{0, \ldots, n-1\} \rightarrow Q \) of automaton \( A \) on a string \( S \) is:
\[
\rho_{A,S}(0) \in \delta(q_0, \lambda_s(0)), q_0 \in Q_0,
\rho_{A,S}(i + 1) \in \delta(\rho_{A,S}(i), \lambda_s(i + 1)).
\]

A run \( \rho_{A,S} \) is accepting iff \( \rho_{A,S}(n-1) \in F \).

An automaton \( A \) accepts a string \( S \) iff there exists an accepting run of \( A \) on \( S \).

The language accepted by automaton \( A \) is \( L(A) = \{ S \in \Sigma^* \mid A \text{ accepts } S \} \). Such a language is called a regular language.

An automaton \( A \) is deterministic iff
\[
\forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| \leq 1,
\]
i.e., for each string \( S \in \Sigma^* \), there is exactly one run \( \rho_{A,S} \). Note that a transition function of a DFA is \( \delta : Q \times \Sigma \rightarrow Q \).

Theorem 2.1 : For any non-deterministic automaton \( A \), there is an equivalent deterministic automaton \( A' \) s.t. \( L(A) = L(A') \).
Proof (Power-set construction):
Suppose $A = (\Sigma, Q, Q_0, F, \delta)$. We construct an equivalent DFA $A'$ as follows:

$A' = (\Sigma, 2^Q, \{X \mid X \subseteq Q_0\}, \{Y \mid Y \cap F \neq \emptyset\}, \delta')$, where $\delta' : 2^Q \times \Sigma \rightarrow 2^Q, \delta'(X, a) = \bigcup_{q \in X} \delta(q, a)$.

3 Regular Languages

A regular expression $e$ has the following syntax.

$e = \emptyset \mid a \mid e' \cup e'' \mid e ; e'' \mid e^* \mid e^+ \mid e^? \mid \bar{e}$ where $a \in \Sigma, e'$ is a regular expression, operator “;” represents concatenation.

Notice sometimes a regular expression can include operator complement: $\bar{e}$. Moreover, $e^+ = e ; e^*$ and $e^? = e \mid \emptyset$. Denote $L(e)$ the language given by $e$.

Theorem 3.1 :

- For any regular expression $e$, there exists a finite automaton $A_e$ s.t. $L(e) = L(A_e)$.
- For any finite automaton $A$, there exists a regular expression $e_A$ s.t. $L(A) = L(e_A)$.

Some complexity results:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-emptiness</td>
<td>$L(A) \neq \emptyset$</td>
</tr>
<tr>
<td>Equivalence</td>
<td>$L(A) = L(A')$</td>
</tr>
<tr>
<td>Containment</td>
<td>$L(A) \subseteq L(A')$</td>
</tr>
<tr>
<td>Universality</td>
<td>$L(A) = \Sigma^*$</td>
</tr>
</tbody>
</table>

Notice: $\text{DLOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \text{NP} \subseteq \text{PSPACE}$.

The closure property of regular languages: the union or intersection of two regular languages is still regular; the complement of a regular language is regular.

4 Logic background

4.1 First-order predicate calculus (FO)

Note: Syntax and Semantics were omitted during the lecture.

4.1.1 Syntax

A vocabulary $\sigma$ has:

1. function symbols: $f_1, f_2, \ldots$ of non-negative arities $m_1, m_2, \ldots$,
2. constant symbols (zero-ary function symbols): $c_1, c_2, \ldots$,
3. predicate symbols: $P_1, P_2, \ldots$ of positive arities $n_1, n_2, \ldots$.

FO terms and FO formulae:
(assume an infinite supply of variables)

1. Each variable $x$ is a term.
2. Free variables (FV) : $x$. 
2. If \( t_1, \ldots, t_k \) are terms and \( f \) is a \( k \)-ary function symbol, then \( f(t_1, \ldots, t_k) \) is a term.
   \( \text{FV} : \bigcup_i \text{FV}(t_i) \).

3. Each constant symbol \( c_i \) is a term.
   \( \text{FV} : \text{none} \).

4. If \( t_1, \ldots, t_k \) are terms and \( P \) is a \( k \)-ary predicate symbol, then \( P(t_1, \ldots, t_k) \) is an atomic formula.
   \( \text{FV} : \bigcup_i \text{FV}(t_i) \).

5. If \( t \) and \( t' \) are terms, then \( t = t' \) is a formula.
   \( \text{FV} : \text{FV}(t) \cup \text{FV}(t') \).

6. If \( \phi \) and \( \psi \) are formulae and \( x \) is a variable, then \( \phi \land \psi, \phi \lor \psi, \neg \phi, \exists x \phi, \text{ and } \forall x \phi \) are formulae.
   \( \text{FV} : \text{FV}(\phi) \cup \text{FV}(\psi), \text{FV}(\phi) \cup \text{FV}(\psi), \text{FV}(\phi) - x, \text{ and } \text{FV}(\phi) - x, \) respectively.

4.1.2 Semantics (Tarski semantics)

An interpretation (or a structure) \( \mathcal{M} \) is:
\[
\mathcal{M} = (U, f^\mathcal{M}_1, f^\mathcal{M}_2, \ldots, c^\mathcal{M}_1, c^\mathcal{M}_2, \ldots, P^\mathcal{M}_1, P^\mathcal{M}_2, \ldots)
\]
where
1. \( U \) is the universe: a non-empty set,
2. \( f^\mathcal{M}_i : U^{n_i} \to U \), where \( n_i \) is the arity of the function symbol \( f_i \),
3. \( c^\mathcal{M}_i \in U \),
4. \( P^\mathcal{M}_i \) of arity \( n_i \) is a subset of \( U^{n_i} \).

Each term \( t(\overline{x}) \), where \( |\overline{x}| = m \), is assigned an element \( t^\mathcal{M}(\overline{x}) \in U \), where \( \overline{x} \in U^m \) is defined by a corresponding object assignment.

For a formula \( \psi \), \( \mathcal{M} \models \psi \), meaning \( \mathcal{M} \) satisfies \( \psi \) (under some object assignment) is defined by:
1. \( \mathcal{M} \models t_1(\overline{x}) = t_2(\overline{x}) \iff t_1^\mathcal{M}(\overline{x}) = t_2^\mathcal{M}(\overline{x}) \),
2. \( \mathcal{M} \models P(t_1(\overline{x}), \ldots, t_k(\overline{x})) \iff (t_1^\mathcal{M}(\overline{x}), \ldots, t_k^\mathcal{M}(\overline{x})) \in P^\mathcal{M} \),
3. \( \mathcal{M} \models \neg \psi \iff \mathcal{M} \not\models \psi \),
4. \( \mathcal{M} \models \psi \land \phi \iff \mathcal{M} \models \psi \) and \( \mathcal{M} \models \phi \),
5. \( \mathcal{M} \models \psi \lor \phi \iff \mathcal{M} \models \psi \) or \( \mathcal{M} \models \phi \),
6. \( \mathcal{M} \models \exists x \psi(\overline{x}, x) \iff \mathcal{M} \models \psi(\overline{x}, b) \) for some \( b \in U \),
7. \( \mathcal{M} \models \forall x \psi(\overline{x}, x) \iff \mathcal{M} \models \psi(\overline{x}, b) \) for every \( b \in U \).
4.2 Representing strings as structures

To represent a string by a structure, we define a vocabulary:

1. a binary predicate symbol $<$, and
2. predicate symbols $P_a, P_b, \ldots$ for every $a, b, \ldots \in \Sigma$.

A string $s \in \Sigma^*$, $|s| = n$, is then represented by the following structure $M_s$:

$$M_s = (U, <^M_s, P^M_s, P^M_{a}, P^M_{b}, \ldots),$$

where $U = \{0, \ldots, n - 1\}$, $<^M_s$ is a binary $<$ on numbers, and

$$P^M_s = \{i | a \text{ occurs at position } i \text{ in } s, i \leq n - 1 \}.$$

For example, consider a string $ababaab$. The structure $M$ corresponding to this string is:

$$((\{0, 1, 2, 3, 4, 5, 6\}, <, P_a, P_b),$$

where $P_a = \{0, 2, 4, 5\}$ and $P_b = \{1, 3, 6\}$.

Consider another example: a string $\Sigma^*a\Sigma^*b\Sigma^*$ where “$a$ occurs before $b$” can be represented by the following FO formula:

$$\exists x, \exists y, x < y \land P_a(x) \land P_b(y).$$

4.3 Monadic second-order logic (MSO)

Syntax and Semantics were omitted in the lecture. Monadic second order logic adds quantification over sets to first-order calculus.

4.3.1 Syntax

We add second-order variables, i.e., variables that range over subsets of the universe, to the vocabulary of FO. We denote second-order variables by capital letters.

MSO terms and MSO formulae:

1. all FO terms and all FO formulae,
2. if $X$ is a second-order variable and $t$ is a term, then $X(t)$ is a formula,
3. if $\psi$ is a formula, then $\exists X \psi$ and $\forall X \psi$ are also formulae.

4.3.2 Semantics

We have an interpretation (or a structure) $M$, as before.

Each term $t(\overline{x})$, where $|\overline{x}| = m$ is assigned an element $t^M(\overline{x}) \in U$, where $\overline{x} \in U^m$ is defined by a corresponding object assignment.

Each formula $\psi(\overline{x}, \overline{S})$, where $|\overline{S}| = m$, is assigned an element $\psi(\overline{x}, \overline{S})$, where $\overline{S} = (S_1, \ldots, S_m), S_i \subseteq U$ is defined by a corresponding object assignment.

For a formula $\psi$, $M \models \psi$, meaning $M$ satisfies $\psi$ (under some object assignment) is defined by:
1. all rules from FO,
2. for $\psi(\pi, X) = X(t(\pi))$, $M \models \psi(\pi, S)$ iff $t^M(\pi) \in S$,
3. $M \models \exists X \psi(X, \ldots)$ iff there exists some $X \subseteq U$ such that $M \models \psi(S, \ldots)$,
4. $M \models \forall X \psi(X, \ldots)$ iff for every $X \subseteq U$, $M \models \psi(S, \ldots)$.

Consider the following example:
Given a regular expression $a^*b^*$, find a sentence $\psi \in MSO$, such that $s \in L(a^*b^*)$ iff $M_s \models \psi$.

Consider the following example:
Given a regular expression $a^*b^*$, find a sentence $\psi \in MSO$, such that $s \in L(a^*b^*)$ iff $M_s \models \psi$.

$$\psi = \exists X, \exists Y, ((\forall x \in X, \forall y \in Y, x < y) \wedge (\forall x, X(x) \vee Y(x))) \wedge$$
$$(\forall x \in X, P_a(x) \wedge \forall y \in Y, P_b(y)))$$

5  B"{u}chi 1959 results

A language $L \subseteq \Sigma^*$ is definable in a logic $L$ (FO or MSO) if there is a sentence $\psi$ of $L$, such that

$$L = \{s \in \Sigma^* \mid M_s \models \psi \} = L(\psi), \ i.e., \ s \in L \Leftrightarrow M_s \models \psi.$$ 

For example, if $\psi$ is the formula from the above example, then $L(\psi) = L(a^*b^*)$.

Theorem 5.1 (B"{u}chi 1960) A language is regular iff it is definable in MSO.

Theorem 5.2 (McNaughton/Pappert 1970) A language is definable in FO iff it is $^*$-free.

Notice a regular expression $e$ is $^*$-free if it is generated by $\emptyset \mid a \mid e \cup e' \mid e; e' \mid \bar{e}$. The Kleene star can often be simulated by other operators. For example, $\Sigma^*a\Sigma^*b\Sigma^* = \emptyset a\emptyset b\emptyset$, whose complement means no $a$ is proceeding $b$.

Proof of B"{u}chi Theorem ($\Rightarrow$):

Here we prove that a regular language is definable in MSO; given a regular language $L \subseteq \Sigma^*$, we need to construct a sentence $\psi \in MSO$, such that $L = L(\psi)$.

Recall that a language is regular iff it is accepted by some deterministic finite automaton.
Let $A = (\Sigma, Q, q_0, F, \delta)$ be a deterministic finite automaton, such that $L(A) = L$.

We now produce $\psi_A$, such that $M_s \models \psi_A$ iff a (unique) run of $A$ on string $s$ ends in an accepting state, i.e., $A$ accepts $s$. Assume that $Q = \{q_0, \ldots, q_{k-1}\}$. Then
\[ \psi_A \equiv \exists X_0, \ldots, X_{k-1} \]
\[ \forall x \left( \bigwedge_{i=0}^{k-1} X_i(x) \land \neg \bigvee_{i \neq j} \exists (X_i(x) \land X_j(x)) \right) \]
\( (X_i \text{'s partition the universe}) \)
\[ \land \forall x \left( \left( \forall y \ x \leq y \rightarrow \bigwedge_{a \in \Sigma} (P_a(x) \rightarrow X_i(x)) \right) \right), \text{ where } q_i = \delta(q_0, a) \]
\( \text{(position 0 is in } X_i, \text{ where } q_i = \delta(q_0, a), \text{ where } a \text{ is the first symbol}) \)
\[ \land \forall x \forall y \left( \left( x < y \land \neg \exists z (x < z \land z < y) \right) \rightarrow \bigwedge_{a \in \Sigma} (X_i(x) \land P_a(y) \rightarrow X_j(y)) \right) \]
\( \text{, where } q_j = \sigma(q_i, a) \)
\( \text{(if position } m \text{ is in } X_i, \text{ the symbol in position } m + 1 \)
\( \text{is } a, \text{ then position } m + 1 \text{ is in } X_j, \text{ where } q_j = \delta(q_i, a)) \)
\[ \land \forall x \left( \forall y (y \leq x) \rightarrow \bigvee_{q_i \in F} X_i(x) \right) \]
\( \text{(if } m \text{ is the last position, then } m \in X_i, \text{ where } q_i \in F) \)

**Proof of } \Leftarrow: \text{ a language definable in MSO is regular. Notice this part was omitted in the lecture.**

**Corollary 5.3** \( MSO = \exists MSO. \)

The proof is directly from the Büchi Theorem since \( MSO \Rightarrow \text{regular language} \Rightarrow \exists \text{MSO.} \)

So far we have used \( \bigvee \) for union, \( \bigwedge \) for intersection, \( \neg \) for complement and \( \exists \) for non-deterministic guess, which leads to powerset construction. Each alternation of \( \forall \) and \( \exists \) causes exponential blowup.

Given \( \varphi \in MSO, \) construct an automaton \( A_\varphi \) s.t. \( L(A_\varphi) = \{ s | M_s \models \varphi \}. \) Then the size of \( A_\varphi \) is greater than or equal to 2 to the \( K \)th power of \( |\varphi|, \) where \( K \) is the number of quantifier alternations in \( \varphi, \) denoted as \( \text{Tower}(|\varphi|, K). \) Notice if there is no quantifier alternations, then \( K = 0. \) The size is \( 2^{|\varphi|}. \) If \( K = 1, \) the size is 2 to the power of \( 2^{|\varphi|}. \) \( \text{Tower}(|\varphi|, K) \) is

\[
2^{2^{|\varphi|}} \left\{ \begin{array}{l}
K \text{ times}
\end{array} \right. 
\]

A function \( f : \mathcal{N} \to \mathcal{N} \) is elementary if there exists a \( K \) s.t. \( \forall n, f(n) \leq \text{Tower}(n, K). \)

**Theorem 5.4 (Meyer, 1979/Stock, 1980)** Suppose there is a function \( f : \mathcal{N} \to \mathcal{N} \) s.t. \( \forall \varphi, \exists A_\varphi, \)
\( \text{s.t. } | A_\varphi | \leq f(| \varphi |). \) Then \( f \) is not elementary, i.e., there is no elementary function to bound the size of \( A_\varphi. \)
6 Trees

A tree can be ranked or unranked. A ranked tree is a \( k \)-ary tree whose node is either a leaf or has \( k \) children. When \( k=2 \), we call the tree a binary tree. There are no restrictions over an unranked tree. An XML document can be represented as an unranked tree.

An XML document can be represented as an unranked tree.

In a binary tree, the tree domain can be represented as a prefix-closed subset of \( \{0,1\}^* \), i.e., Figure 1. A tree \( T = (D, \lambda_T) \) where \( D \) is the tree domain and \( \lambda_T : D \rightarrow \Sigma \) is a function that maps each position to a symbol in the alphabet. A string is 1-ary tree whose domain is the prefix-closed subset of \( 0^* \), i.e., \( \{\epsilon, 0, 00, \ldots\} \).

6.1 Bottom-Up Tree Automata

A non-deterministic bottom-up tree automaton can be represented as a tuple \( A = (Q, Q_0, F, \delta) \), where \( Q \) is a finite set of states, \( Q_0 \subseteq Q \) is a set of initial states, \( F \subseteq Q \) is the set of final (accepting) states and \( \delta : Q \times Q \times \Sigma \rightarrow 2^Q \) is the transition function. For example, say a node \( a \) has two leaves. Suppose the automaton is in states \( q_1 \) and \( q_2 \) at the leaves, respectively. Then a state \( q \) at \( a \) is in \( \delta(q_1, q_2, a) \) (Figure 2).

A run of a tree automaton \( A \) over a tree \( T \) is \( \rho_{A,T} : D \rightarrow Q \) s.t.

1. if \( s, s0, s1 \in D \), then \( \rho_{A,T}(s) \in \delta(\rho_{A,T}(s0), \rho_{A,T}(s1), \lambda_T(s)) \).
2. if \( s \) is a leaf, then \( \rho_{A,T}(s) \in \delta(q_0, q_0, \lambda_T(s)) \) for some \( q_0 \in Q_0 \) (Figure 2).

A run is successful if \( \rho_{A,T}(\epsilon) \in F \). A tree \( T \) is accepted by \( A \) if there is a successful run \( \rho_{A,T} \) on \( T \). \( L(A) = \{ T \mid T \text{ accepted by } A \} \).

Automaton \( A \) is deterministic if \( | \delta(q, q', a) | \leq 1 \) for all \( q, q', a \). Each bottom-up tree automaton is equivalent to a deterministic one (proof by powerset construction).

6.2 Top-Down Tree Automata

A non-deterministic top-down tree automaton can be represented as a tuple \( A = (Q, Q_0, F, \delta) \), where \( Q \) is a finite set of states, \( Q_0 \subseteq Q \) is a set of initial states, \( F \subseteq Q \times \Sigma \) is the set of pairs of final (accepting)
states and symbols in Σ, and δ : Q × Σ → 2^Q × Q is the transition function.

For every non-deterministic top-down tree automaton, there is an equivalent non-deterministic bottom-up tree automaton, and vice versa. However, the set of deterministic top-down tree automata-
tons is properly contained in the set of non-deterministic top-down tree automatons.

**Proof**: let a language L contain only two trees as in Figure 3. Suppose there is a deterministic tree automaton A whose language is L. The states of the trees are shown in Figure 4. Notice the pairs of states with leaf symbols are all in F since L is the language of A. Now that the pairs (q₁, a) and (q₂, a) are in F, the third tree in Figure 4 should also be in L as all pairs of its states with leaf symbols are in F. In other words, this tree is accepted by A but not in L. Contradiction! Hence there does not exist such an A.

### 6.3 Logics for Trees

We represent a tree T = (D, λ_T) as a first-order structure

\[ \mathcal{M}_T = (D, <_0, <_1, (P_a)_{a \in \Sigma}) \]

of vocabulary σΣ expanded with two binary relations <₀ and <₁. Relation <₀ represents left successor, i.e., s <₀ s₀ and <₁ represents right successor, i.e., s <₁ s₁. P_a is interpreted as \{ s ∈ D | λ_T(s) = a \},

\[ 8 \]
i.e., the set of nodes labeled $a$.

In MSO, to express “$y$ is a descendant of $x$”, we use $\varphi(x, y)$ as follows.

$$\exists X, \ (X(x) \land X(y) \land \ (\forall z, z \text{ is the parent of } x \rightarrow \neg X(z)) \land \ (\forall u \in X, u \text{ is a leaf } \lor (\text{exactly one of } u_0, u_1 \in X, \text{ where } u <_0 u_0, u <_1 u_1)).$$

Notice if we would like to use FO to describe similar statements on trees, we need $<_0, <_1$ and $<_*$ as descendant successor.

We call $X$ a cut of the tree if every path from the tree root to a leaf intersects $X$.

Given $\varphi(X)$, represent it as a circuit with $X$ as a set of leaves and $\lor, \land, \neg$ as internal nodes. Then $\varphi(X)$ is true iff the circuit evaluates to be 1.

**Definition 6.1** A tree language $L$ (set of binary trees) is regular if it is accepted by a (bottom-up) tree automaton.

**Definition 6.2** A tree language $L$ is $L$-definable if there is a sentence $\varphi$ of $L$ s.t. $\forall T \in L \iff T \models \varphi$.

**Theorem 6.1** (Thatcher/Wright, 1970) A tree language is regular iff it is MSO-definable/$\exists$MSO-definable.

There is no elementary function to bound the complexity of translation of MSO to automaton. The reason is the same as the previous one in Section 5.

**Some complexity results:**

<table>
<thead>
<tr>
<th>Non-emptiness</th>
<th>$L(A) \neq \emptyset$</th>
<th>PTIME-complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalence</td>
<td>$L(A) = L(A')$</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>Containment</td>
<td>$L(A) \subseteq L(A')$</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>Universality</td>
<td>$L(A)$ is the set of all trees</td>
<td>EXPTIME-complete</td>
</tr>
</tbody>
</table>