CS 2428 - Logic and Automata

Lecture #2: 20 January 2005

Lecturer: Leonid Libkin

Scribe Notes by: Shiva Nejati

1 *MSO* on strings and regular languages

Theorem 1 (Buchi 1959) *A language is regular if and only if it is definable in MSO.*

In the previous lecture, we presented the proof for the "if" part of Theorem 1. That is, we proved that for every regular language L there exists a sentence $\varphi \in MSO$ such that

$$\mathfrak{M}_s \models \varphi \Leftrightarrow s \in L$$

where $\mathfrak{M}_s = (\{0, 1, \ldots, n\}, <, (P_a)_{a \in \Sigma})$ is the first-order structure corresponding to a string $s \in L$.

Example Given a string $s = abaab$, the corresponding first-order structure is

$$\mathfrak{M}_s = (\{0, 1, 2, 3, 4\}, <, P_a = \{0, 2, 3\}, P_b = \{1, 4\})$$

We now prove the "only if" part of Theorem 1. Suppose we are given an MSO sentence φ and want to construct an NFA $\mathcal{A}_\varphi$ such that $L(\mathcal{A}_\varphi) = L(\varphi)$ where $L(\varphi) = \{s \in \Sigma^* \mid \mathfrak{M}_s \models \varphi\}$. The main idea is to proceed by structural induction on MSO formulae. Note that we cannot use induction on sentences because sentences do not have any free variables. Therefore, we prove the claim by induction on the construction of MSO formulae instead of MSO sentences.

First we show how to eliminate the first-order variables x_i and get equivalent formulas containing only set variables. To this end, we introduce the following atomic MSO_0 formulas:

- $X \subseteq Y$ and $X \subseteq P_a$
- $Sing(X)$ (X is a singleton set)
- $X < Y$ ($X = \{x\}$, $Y = \{y\}$ and $x < y$)

Every MSO formula can be converted into an MSO_0 formula by replacing every first-order variable with a second-order variable and using the above atomic MSO_0 formulae.

Fact $MSO = MSO_0$

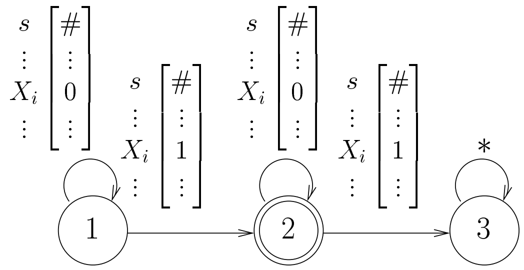
Therefore, to prove the claim it suffices to consider MSO_0 formulae.

We prove by induction that every formula $\varphi(X_1, \dots, X_n)$ can be translated to an automaton $\mathcal{A}_\varphi$ over the alphabet $\Sigma \times \{0,1\}^n$. The input strings of $\mathcal{A}_\varphi$ are matrices of the following form:

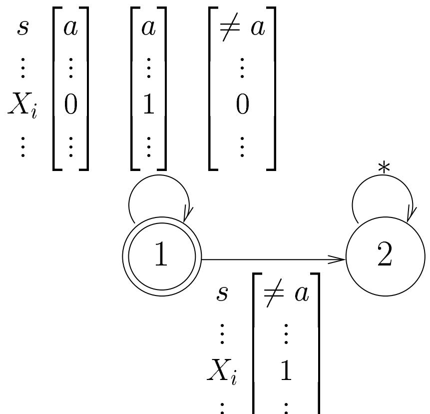
$$\begin{array}{c} s \\ X_1 \\ \vdots \\ X_n \end{array} \begin{bmatrix} a & b & a & b \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

The first row of the matrix is a string $s \in \Sigma^*$ and the rest of the rows are sets of positions corresponding to monadic variables X_i's. We want to construct an automaton $\mathcal{A}_\varphi$ so that $\mathcal{A}_\varphi$ accepts $\begin{bmatrix} s \\ X_1 \\ \vdots \\ X_n \end{bmatrix}$ if and only if $\mathfrak{M}_s \models \varphi(X_1, \dots, X_n)$. For each $\varphi(X_1, \dots, X_n)$, we inductively construct a finite automaton over $\Sigma \times \{0,1\}^n$. For the inductive hypothesis we consider the atomic MSO_0-formulae:

- $Sing(X_i)$:



- $X_i \subseteq P_a$:



Constructing the automata for the rest of atomic MSO_0-formulae is straightforward.

For the inductive step, we suppose that for the formulae φ_1 and φ_2 the corresponding NFA's $\mathcal{A}_1$ and $\mathcal{A}_2$ are given. We only consider the connectors $\vee$, $\neg$, and the quantifier $\exists$, because the rest can be expressed as a combination of these connectives:

- $\varphi = \varphi_1 \vee \varphi_2$: given automata $\mathcal{A}_1$ and $\mathcal{A}_2$, it is possible to construct the union automaton that recognizes $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

- $\varphi = \neg\varphi$: given an NFA $\mathcal{A}_1$, it is possible to construct the corresponding DFA using power-set construction on $\mathcal{A}_1$, and then, in the resulting DFA, we declare all states $Q \setminus F$ as accepting states and all F states as rejecting states to obtain the complement automaton.
- $\varphi(X_1, \ldots, X_n) = \exists X_{n+1}\psi(X_1, \ldots, X_{n+1})$: given an NFA $\mathcal{A}_\psi = (Q, \Sigma \times \{0,1\}^{n+1}, \delta_\psi, F)$ that recognizes $L(\psi)$, we can construct an NFA $\mathcal{A}_\varphi = (Q, \Sigma \times \{0,1\}^n, q_0, \delta_\varphi, F)$ that recognizes $L(\varphi)$, where $\delta_\varphi(q, \alpha) = \delta_\psi(q, \begin{bmatrix}\alpha\\0\end{bmatrix}) \cup \delta_\psi(q, \begin{bmatrix}\alpha\\1\end{bmatrix})$ for $\alpha \in \Sigma \times \{0,1\}^n$.

  At each step the automaton $\mathcal{A}_\varphi$ guesses whether the input corresponding to X_{n+1} is 0 or 1. It can be seen that $\mathcal{A}_\varphi$ accepts a matrix $s \in (\Sigma \times \{0,1\}^n)^*$ if and only if there exists a witness for X_{n+1}.

Definition Existential *MSO* (or *EMSO*, or monadic Σ^1_1) is defined as the restriction of *MSO* that consists of the formulae of the form:

$$\exists X_1 \ldots \exists X_n \varphi(X_1, \ldots, X_n)$$

where φ is first-order formula without any second-order quantification.

In the "only if" part of the proof of Theorem 1, we showed that over strings every *MSO* sentence can be represented by an automaton. In addition, in the "if" part of the proof, it was shown that the behavior of every automaton can be captured by an *EMSO* sentence. Hence, we obtain the following:

Corollary 2 *Over strings MSO = EMSO.*

Moreover, it can be shown that over strings a fragment of *EMSO* where at most one monadic existential quantifier is allowed is equivalent to *MSO*.

Theorem 3 *Over strings MSO is equivalent to a fragment of EMSO where only one monadic existential quantifier is allowed.*

1.1 Complexity of *MSO*

So far, we showed the transformation of an *MSO* formula into an equivalent automaton and vice versa. Now, we focus on the complexity of this transformation. From the proof, it can be seen that the complexity of transforming an automaton into an *MSO* sentence is linear in the size of the automaton.

In the transformation of an *MSO* formula into an automaton, the most expensive step is the translation of negation. This is because for negation, we have to go from an NFA to a DFA using power-set construction which entails an exponential blow-up. For example, consider an *MSO* sentence ϕ of the form:

$$\phi = (\exists \ldots \exists)(\forall \ldots \forall)(\exists \ldots \exists) \ldots \varphi$$

We see at each quantifier alternation, one need to make the automaton deterministic. Hence, the size of the resulting automaton will be bounded by (roughly)

$$\left.2^{2^{\cdot^{\cdot^{2^n}}}}\right\} k \text{ times}$$

where n is the size of the automaton corresponding to φ, and k is the number of alternations of quantifiers. The size of the automaton is actually *non-elementary* in terms of $||\phi||$. We recall that a function $f : \mathbb{N} \rightarrow \mathbb{N}$ is elementary if for a fixed k,

$$f(n) \ < \ \left.2^{2^{\cdot^{\cdot^{2^n}}}}\right\}k \text{ times} \quad \text{for all } n.$$

In fact, it is known that for converting *MSO* formulae into automata, we cannot hope for a better bound.

Theorem 4 (Meyer,Stockmeyer 1971) *There is no translation of MSO-formulae φ to automata in which the number of states of the resulting automaton can be bounded by a function of the form*

$$\left.2^{2^{\cdot^{\cdot^{2^n}}}}\right\}k \textit{ times}$$

with a constant k.

1.2 Automata-based model-checking

Theorem 1 has interesting practical applications in model-checking. The model-checking problem is the following: given a transition system and a specification, we want to determine whether the transition system satisfies the specification. The transition system can be seen as an automaton, and the specification is typically expressed in terms of some logical formula. So the model-checking problem can be restated as follows: given a finite transition system represented by an NFS $\mathcal{A}_p$ and a specification formulated as an *MSO*-formula φ, does every word accepted by $\mathcal{A}_p$ satisfy the specification φ?

To solve the model-checking problem we introduce error scenarios. A string s is an error scenario if $s \in L(\mathcal{A}_p)$ but $\mathfrak{M}_s \not\models \varphi$. That is, $s \in L(\mathcal{A}_p) \cap L(\mathcal{A}_{\neg\varphi})$, or $s \in L(\mathcal{A}_p \times \mathcal{A}_{\neg\varphi})$.

So, given an automaton $\mathcal{A}_p$ and an *MSO*-formula φ, to solve the model checking problem, we have to do the following:

1. Construct $\mathcal{A}_{\neg\varphi}$.
2. Construct the product automaton $\mathcal{A}_p \times \mathcal{A}_{\neg\varphi}$.
3. Check whether $L(\mathcal{A}_p \times \mathcal{A}_{\neg\varphi}) \neq \emptyset$.

The complexity of steps 2 and 3 is polynomial time, whereas the complexity of step 1 is exponential. This approach, although very expensive, reveals an interesting connection between logic, automata, and model-checking algorithms. This approach to model-checking is known as automata-based model-checking.

1.3 An alternative proof

The proof that we gave for the "only if" part of Theorem 1 was an inductive proof based on converting *MSO*-sentences into automata directly. Here, we give an alternative model-theoretic proof that does not use induction. For this proof, we need to define quantifier rank and *MSO* rank-k type and provide some background information.

Definition The quantifier rank of a formula $\mathsf{qr}(\varphi)$ is its depth of quantifier nesting, that is:

- If φ is atomic, then $\mathsf{qr}(\varphi) = 0$.
- $\mathsf{qr}(\varphi_1 \vee \varphi_2) = \mathsf{qr}(\varphi_1 \wedge \varphi_2) = \max(\mathsf{qr}(\varphi_1), \mathsf{qr}(\varphi_2))$.
- $\mathsf{qr}(\neg\varphi_1) = \mathsf{qr}(\varphi_1)$.
- $\mathsf{qr}(\exists\, \varphi) = \mathsf{qr}(\forall\, \varphi) = \mathsf{qr}(\varphi) + 1$.

Definition Given a structure $\mathfrak{M}$, the *MSO* rank-k type is defined as

$$tp^k_{MSO}(\mathfrak{M}) = \{\varphi \in MSO \mid \mathsf{qr}(\varphi) \leq k, \mathfrak{M} \models \varphi\}$$

Fact Up to logical equivalence, there are finitely many formulae of quantifier rank-k.

A set of *MSO* sentences of quantifier rank-k is a rank-k type if it is of the form $tp^k_{MSO}(\mathfrak{M})$.

Now, we are ready to give the proof. Let $\tau_0, \tau_1, \ldots, \tau_n$ enumerate all the rank-k types of structures $\mathfrak{M}_s$. Let $\tau_0 = tp^k_{MSO}(\mathfrak{M}_\epsilon)$. Let ψ be an *MSO*-sentence of quantifier rank-k. We define an automaton

$$\mathcal{A}_\psi = (Q = \{\tau_0, \ldots, \tau_k\}, q_0 = \tau_0, F, \delta_\psi)$$

where $F = \{\tau_i \mid \exists s\; tp^k_{MSO}(\mathfrak{M}_s) = \tau_i, \mathfrak{M}_s \models \psi\}$, and the transition function δ_ψ is defined as

$$\tau_j \in \delta_\psi(\tau_i, a) \Leftrightarrow \exists s\; tp^k_{MSO}(\mathfrak{M}_s) = \tau_i \text{ and } tp^k_{MSO}(\mathfrak{M}_{sa}) = \tau_j$$

The following theorem completes the proof. For a detailed proof, refer to [1] Chapter 7.

Theorem 5

1. $s \in L(\mathcal{A}_\psi) \Leftrightarrow \mathfrak{M}_s \models \psi$
2. $\mathcal{A}_\psi$ *is a DFA.*

2 FO on strings and star-free languages

So far, we discussed that *MSO* on strings captures regular languages. Now the question is that what can be said about the the class of languages captured by *FO*? It turns out that *FO* corresponds to a well-known class of languages, that we define below.

Definition A star-free regular expression over Σ is an expression built from the symbols $\emptyset$ and a, for each a in Σ, using the operations of union $+$, complement $\bar{\ }$, and concatenation $\cdot$. Such a regular expression e denotes a language $L(e)$ over Σ as follows:

- $L(\emptyset) = \emptyset; L(a) = \{a\}$ for $a \in \Sigma$.
- $L(e_1 + e_2) = L(e_1) \cup L(e_2)$.
- $L(\bar{e}) = \Sigma^* - L(e)$.
- $L(e_1 \cdot e_2) = \{s_1 \cdot s_2 \mid s_1 \in L(e_1), s_2 \in L(e_2)\}$.

A language denoted by a star-free expression is called a star-free language.

Note that some of the regular expressions that use the Kleene star $*$ are actually star-free. For example, Σ^* is star-free because $\Sigma^* \equiv \bar{\emptyset}$.

Theorem 6 *A language is star-free if and only if it is definable in FO.*

Consider a DFA $\mathcal{A} = (Q, q_0, F, \delta)$. For each string $s \in \Sigma^*$, we define a function $f_s : Q \to Q$ such that $f_s(q) = q'$ if when starting from q with input s, we arrive at state q'. Note that the set $\{f_s \mid s \in \Sigma^*\}$ is finite. Given an automaton $\mathcal{A}$, we define a monoid $\mathcal{M}(\mathcal{A}) = ((f_s)_{s\in\Sigma^*}, f_\epsilon, \circ)$, where the monoid operation $\circ$ is function composition. That is, $f_s \circ f_{s'} = f_{s's}$ for every $s, s' \in \Sigma^*$.

Recall that a monoid is a tuple $(M, \epsilon, \circ)$, where $\circ$ is an associative binary operation, and ϵ is the identity element.

Definition A monoid $(M, \epsilon, \circ)$ is aperiodic if there exists k such that for all $x \in M$, $x^k = x^{k+1}$, where $x^k = \overbrace{x \circ \ldots \circ x}^{k \text{ times}}$.

Theorem 7 *A regular language is $*$-free iff for its minimal automaton $\mathcal{A}$, $\mathcal{M}(\mathcal{A})$ is aperiodic.*

Corollary 8

- *It is decidable if a regular language is $*$-free.*
- *It is decidable if an MSO sentence over strings is equivalent to an FO sentence.*

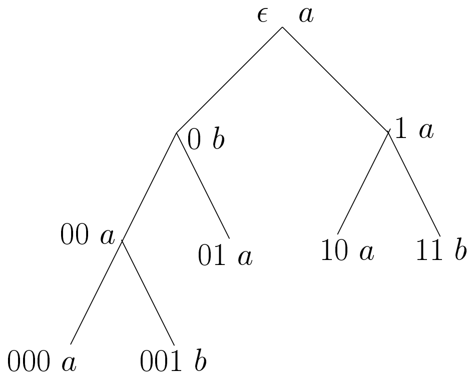
3 Trees and tree automata

We now move from strings to trees.

Definition A (binary) tree domain is a prefixed-closed subset D of $\{0,1\}^*$ such that if $s \in D$, then either

- both $s0, s1 \in D$.
- neither $s0, s1 \in D$.

A Σ-tree (Σ-labeled tree) T is a pair (D, λ) where D is a tree domain and λ is a function from D to Σ (the labeling function).



Definition A nondeterministic bottom-up tree automaton (NTA) is a tuple $\mathcal{A} = (Q, q_0, F, \delta : \Sigma \times Q \times Q \to 2^Q)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final (accepting) states, and $\delta : \Sigma \times Q \times Q \to 2^Q$ is the transition function. A run of NTA $\mathcal{A}$ on a tree $T = (D, \lambda)$ is a function $\rho : D \to Q$ such that

1. if $s \in D$ is a leaf labeled a, then $\rho(s) \in \delta(a, q_0, q_0)$.

2. if $s \in D$, $s0, s1 \in D$, and $\lambda(s) = a$, then $\rho(s) \in \delta(a, \rho(s0), \rho(s1))$.

A run is accepting if $\rho(\epsilon) \in F$ (the root is in the accepting states). A tree T is accepted by an NTA $\mathcal{A}$ iff for T there is an accepting run. $L(A)$ is the set of all trees accepted by $\mathcal{A}$. Sets of the form $L(\mathcal{A})$ are called regular tree languages.

A tree automaton is deterministic if δ is of the form $\Sigma \times Q \times Q \to Q$. Each DTA has exactly one run on every tree.

Fact For each NTA, there exists an equivalent DTA.

The proof is exactly the same as the power-set construction proof for word automata.

Like strings, trees can be captured by first-order structures. Each tree can be viewed as a first-order structure $(D, <_0, <_1, (P_a)_{a \in \Sigma})$, where D is a set of nodes, $<_i$ is $\{(s, s \cdot i) \mid s, s \cdot i \in D\}$ for $i \in \{0, 1\}$, and P_a is interpreted as $\{s \mid \lambda(s) = a\}$.

Theorem 9 (Thatcher, Wright, Doner 1968-1970) *A set of trees is definable in MSO iff it is regular.*

The proof is completely analogous to the one of Theorem 1.

References

[1] L. Libkin, *Elements of Finite Model Theory*, Springer-Verlag, 2004.