MSO on Words

Mahesh Viswanathan

Fall 2018

Monadic Second Order (MSO) Logic is the fragment of second order logic where relational variables in a formula are restricted to be of arity 1. We will now study the expressive power of MSO and decision problems for MSO when restricted to word structures. The main result, due to Büchi, Elgot, and Trakhtenbrot, we will prove is that the collection of words is definable in MSO if and only if it is regular. We begin by recalling regular languages and their properties.

1 Finite Automata and Regular Languages

Finite automata are the simplest computational model. They describe algorithms that solve a decision problems using only finite memory. An example finite automaton is shown in Figure 1. It has a finitely many states (shown as vertices in Figure 1), and transitions from one state to another on reading a symbol from the input (shown as labeled directed edges in Figure 1). The automaton is assumed to start in a special state called the *initial* or *start* state; this is indicated by an incoming arrow with no source in Figure 1. In each step, the automaton reads the next symbol from the input, and moves to a new state by taking a transition (edge) labeled by the input symbol read. After reading the entire symbol, if the state of the automaton is a *final* or *accepting* state (shown as double circled vertices in Figure 1) then the input is accepted; otherwise, the input is rejected. The formal definition is as follows.

Definition 1. A *(nondeterministic) finite automaton (NFA)* is $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of (control) states
- Σ is finite (input) alphabet
- $\delta : Q \times \Sigma \to 2^Q$ is transition function
- $q_0 \in Q$ is initial state
- $F \subseteq Q$ the set of final/accepting states

If for every $a \in \Sigma$ and $q \in Q$, $|\delta(q, a)| = 1$ then M is said to be *deterministic*.

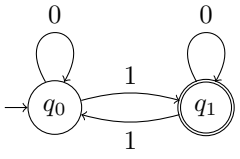

Figure 1: Example Finite Automaton

On any sequence of symbols, the NFA when started from the initial state, could have different computations, based on the choices made in each step. It is convenient to describe the effect of running the automaton on a sequence of symbols, by identifying the set of states it is could possibly be in. This can be defined inductively as follows.

Definition 2. For an NFA $M = (Q, \Sigma, \delta, q_0, F)$, string w, and state $q_1 \in Q$, $\hat{\delta}(q_1, w)$ is the set of all states M could be in after reading w. That is,

$$\hat{\delta}(q_1, w) = \begin{cases} \{q_1\} & \text{if } w = \varepsilon \\ \cup_{q \in \hat{\delta}(q_1, u)} \delta(q, a) & \text{if } w = ua \end{cases}$$

An input string is *accepted* by an automaton, if one of the states it could be in after reading the input is a final or accepting state. The language recognized by an automaton (or the decision problem solved by the automaton) is the set of inputs that are accepted by it. This is formally defined as follows.

Definition 3. The language *recognized* by finite automaton M is $\mathbf{L}(M) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.

Deterministic finite automata (DFA) are special finite automata with the property that from any state, on any input symbol, there is a unique next state it could transition to (see Definition 1). Figure 1 shows an example of such an automaton. Thus, if a language A is recognized by a DFA (say) M, then A is also recognized by an NFA, namely M itself. The converse, though true, is not that immediate and is an important result in automata theory.

Theorem 4 (Rabin-Scott)**.** *For any NFA M, there is a DFA $\mathsf{dfa}(M)$ such that $\mathbf{L}(M) = \mathbf{L}(\mathsf{dfa}(M))$.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$. The DFA $\mathsf{dfa}(M)$ is obtained by what is often called a "subset construction". The states of $\mathsf{dfa}(M)$ corresponds to sets of states of M, and $\mathsf{dfa}(M)$ "simulates" M by tracking the set of states M could be in. Formally, $\mathsf{dfa}(M) = (2^Q, \Sigma, \delta', \{q_0\}, F')$ where $F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$ and $\delta'(S, a) = \cup_{q \in S} \delta(q, a)$. □

1.1 Regular Languages

An important class of languages, called *regular languages*, is defined inductively from simple languages using algebraic operations on languages. The class of languages recognized by finite automata is the same as the class of regular languages; this is Kleene's theorem that we will establish at the end of this section. We begin by defining regular languages.

Definition 5. For a pair of languages $A, B \subseteq \Sigma^*$, their *concatenation* is defined as $AB = \{uv | u \in A \text{ and } v \in B\}$. A^i is defined to be i-fold concatenation of A with itself, i.e., $A^i = \{u_1 u_2 \cdots u_i | u_1, u_2, \ldots u_i \in A\}$. Observe that $A^1 = A$. By definition, we take $A^0 = \{\varepsilon\}$ for any A.

The *Kleene closure* of a language A is defined to be

$$A^* = \bigcup_{i \in \mathbb{N}} A^i.$$

Regular expressions are inductively defined expressions that describe languages. They are built from simple languages like $\emptyset$, $\{\varepsilon\}$ and $\{a\}$ (where $a \in \Sigma$) using (set) union, concatenation, and Kleene closure. Their syntax and semantics is defined as follows.

Definition 6. *Regular expressions*, over alphabet Σ, are inductively defined as follows.

- $\emptyset$, ε, and a are regular expressions, where $a \in \Sigma$.
- If r and s are regular expressions, then $(r + s)$, (rs) and (r^*) are also regular expressions.

Each regular expression defined a set of strings over Σ. This can be defined inductively, in the same manner in which we defined the expressions.

- $\mathbf{L}(\emptyset) = \emptyset$, $\mathbf{L}(\varepsilon) = \{\varepsilon\}$, and $\mathbf{L}(a) = \{a\}$.
- $\mathbf{L}((r + s)) = \mathbf{L}(r) \cup \mathbf{L}(s)$
- $\mathbf{L}((rs)) = \mathbf{L}(r)\mathbf{L}(s)$
- $\mathbf{L}((r^*)) = (\mathbf{L}(r))^*$

A language A is *regular* if there is a regular expression r such that $\mathbf{L}(r) = A$.

Kleene proved that regular languages are exactly those that are recognized by finite automata.

Theorem 7 (Kleene)**.** *A is regular if and only if there is a finite automata M such that $A = \mathbf{L}(M)$.*

The proof of Theorem 7 is nontrivial but can be found in any standard textbook on theory of computation. We skip it here. The consequence of Kleene's theorem and Theorem 4 is that DFAs, NFAs, and regular expressions have the same expressive power.

1.2 Properties of Regular Languages

The class of regular languages are closed under a number of important algebraic operations.

Theorem 8. *Regular languages are closed under all Boolean operations. That is, if A and B are regular languages then so are $\overline{A}$, $A \cup B$, and $A \cap B$.*

Proof. If A is regular than, by Theorem 7 and Theorem 4, we have that there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $\mathbf{L}(M) = A$. It is easy to see that the DFA $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ recognizes $\overline{A}$.

If A and B are regular then (by definition) there are regular expressions r_A and r_B such that $\mathbf{L}(r_A) = A$ and $\mathbf{L}(r_B) = B$. Then (again by definition) $r_A + r_B$ is regular expression defining $A \cup B$, and to $A \cup B$ is regular.

Finally, since $A \cap B = \overline{\overline{A} \cup \overline{B}}$ and regular languages are closed under complementation and union, they are also closed under intersection. □

Definition 9. A function $h : \Sigma^* \to \Delta^*$ is a *homomorphism* iff $h(\varepsilon) = \varepsilon$ and for all $u, v \in \Sigma^*$, $h(uv) = h(u)h(v)$.

Let us fix a homomorphism $h : \Sigma^* \to \Delta^*$. The *homomorphic image* of a language $A \subseteq \Sigma^*$, is $h(A) = \{h(u) \mid u \in A\}$. The *inverse homomorphic image* of a language $B \subseteq \Delta^*$ is $h^{-1}(B) = \{v \mid h(v) \in B\}$.

It is important to observe that h and h^{-1} are not functional inverses of each other, when applied to languages. For example, consider the homomorphism $h : \{a\}^* \to \{0, 1\}^*$ that maps all strings to ε, i.e., $h(w) = \varepsilon$ for all $w \in \{a\}^*$. Observe that $h(\{a\}) = \{\varepsilon\}$. However, $h^{-1}(\{emptystr\}) = \{a\}^*$. Regular languages are closed under homomorphic and inverse homomorphic images.

Theorem 10. *For a homomorphism h and regular language A, $h(A)$ and $h^{-1}(A)$ are regular.*

Proof. Let us fix $h : \Sigma^* \to \Delta^*$ to be a homomprhism. For a regular expression r, we can define the "homomorphic image" of r as another regular expression defined inductively as follows.

- $\hat{h}(\emptyset) = \emptyset$, $\hat{h}(\varepsilon) = \varepsilon$, and $\hat{h}(a) = h(a)$.
- $\hat{h}(r + s) = \hat{h}(r) + \hat{h}(s)$
- $\hat{h}(rs) = \hat{h}(r)\hat{h}(s)$
- $\hat{h}(r^*) = (\hat{h}(r))^*$

One can prove by induction that $\mathbf{L}(\hat{h}(r)) = h(\mathbf{L}(r))$. This establishes that regular languages are closed under homomorphic images.

Let $M = (Q, \Delta, \delta, q_0, F)$ be a DFA. Consider the DFA $h^{-1}(M) = (Q, \Sigma, \delta', q_0, F)$ where the transition function δ' is given by

$$\delta'(q, a) = \hat{\delta}(q, h(a)).$$

One can prove that $\mathbf{L}(h^{-1}(M)) = h^{-1}(\mathbf{L}(M))$. This establishes the closure of regular languages under inverse homomorphic images. □

We conclude this section by observe that the mebership problem and emptiness problem for finite automata are decidable in linear time.

Theorem 11. *The following problems are decidable in linear time.*

Membership *Given an automaton M and string u, determine if $u \in \mathbf{L}(M)$.*

Emptiness *Given an automaton M, determine if $\mathbf{L}(M) = \emptyset$.*

Proof. The membership problem can be decided by iteratively determining the set of states of M as it reads the symbols of u. This will take time that is proportional to the length of u (and the number of states/transitions of M).

The emptiness problem can be decided by determining if there is a path from the initial state to some final state. This requires running a DFS/BFS on the graph representation of the automaton. □

2 Büchi-Elgot-Trakhtenbrot Theorem

The main result we will establish in this section is that the set of words expressible in MSO is exactly the collection of regular languages. Recall that there is a 1-to-1 and onto mapping between elements of Σ^* and word structures over Σ. Hence we will interchangable use w to denote both an element of Σ^* and the corresponding word structure. Recall that the truth of a MSO sentence in a word structure does not depend on the assignment of values to variables and relational variables. Therefore, we can talk about the sets of words that can be defined in MSO.

Definition 12. A set of word structures A over alphabet Σ is said to be definable in MSO, if there is a MSO sentence φ over signature $\tau_W = (<, S, \{Q_a\}_{a \in \Sigma})$ such that

$$A = \{w \mid w \models \varphi\}$$

For a sentence φ, we will denote by $[\![\varphi]\!]$ the set $\{w \mid w \models \varphi\}$.

Theorem 13 (Büchi-Elgot-Trakhtenbrot)**.** *A set of words A is definable in MSO if and only if A is regular.*

There are two directions to establishing this result, and we will consider them in order.

2.1 Regular Languages are Definable

In this section, we will show how for any regular language A, we can construct a MSO sentence φ_A that defines the set A. The construction of the MSO sentence will be based on a finite automaton recognizing the regular language A. Let us look at an example.

Example 14. Consider the example finite automaton shown in Figure 2. It recognizes the collection of all strings containing consecutive bs. This can be easily expressed in MSO as the set of strings having successive positions that are labeled by b. Thus, the desired MSO sentence is

$$\exists x \exists y (Q_b x \wedge Sxy \wedge Q_b y)$$

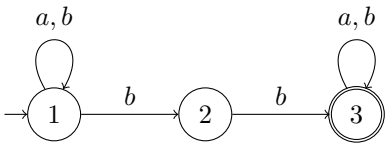

Figure 2: Finite automaton recognizing $(a+b)^*bb(a+b)^*$

Example 14 constructs a regular expression that relies on understanding the language recognized by the automaton. To construct such an MSO sentence for an arbitrary automaton, we will instead rely on a different approach. It will be similar to the proofs of the Cook-Levin theorem, Church-Turing theorem, and Gödel's incompleteness theorem, where we try to encode the conditions for the existence of an accepting computation in logic. Effectively, the sentence will say that there is an accepting computation of the automaton on the word encoded as a structure. We will describe this construction for the example automaton shown in Figure 2, rather than describe the translation in general.

The idea behind the construction is to say that the automaton has an accepting computation on the given word structure. Recall that a computation of an automaton is a sequence of states of the automaton. Sequences like this can be captured by a sets corresponding to each state, that capture the positions where the automaton is in that particular state. For example, consider the input word *abbab*. The automaton in Figure 2 has an accepting run which is 112333. This computation will be captured by 3 sets — X_1, X_2, X_3 — which will contain the positions where the automaton is in state 1, 2, and 3, respectively. Thus, the run 112333 corresponds to the sets $X_1 = \{1, 2\}$, $X_2 = \{3\}$, and $X_3 = \{4, 5\}$. Notice that the last state reached is not explicitly in any set; this is because the word (as a structure) only has positions $\{1, 2, 3, 4, 5\}$, and position "6" is not an element of the universe.

We just observed that computations of an automaton on a word can be captured by sets (or unary relations). The desired formula needs to say "the word has an accepting computation" which means it has to say that "there is an encoding of a sequence of states such that"

- "The encoding is that of a sequence of states"
- "The first state is the initial state"
- "Each state follows from the previous state by a transition"
- "On reading the last symbol, from the last state, we get to an accepting state"

Let us describe how these conditions can be encoded. For the example in Figure 2, our sentence will take the following form.

$$\begin{aligned}\varphi = \exists X_1 \exists X_2 \exists X_3 \quad & \text{exactly one state in each position} \wedge \\ & \forall x(\mathsf{first}(x) \to X_1 x) \wedge \\ & \forall x \forall y(Sxy \to \text{state at } y \text{ follows from state at } x \\ & \qquad \text{by transition}) \wedge \\ & \forall x(\mathsf{last}(x) \to \text{final state reached from state at } x \\ & \qquad \text{by reading symbol at } x)\end{aligned}$$

Let us see how to encode each of the english phrases in the above sentence.

- "Exactly one state in each position"

$$\forall x((X_1 x \vee X_2 x \vee X_3 x) \wedge \neg(X_1 x \wedge X_2 x) \wedge \neg(X_1 x \wedge X_3 x) \wedge \neg(X_2 x \wedge X_3 x))$$

- "State at y follows from State at x by transition"

$$\begin{array}{ll} & (X_1x \wedge Q_ax \wedge X_1y) \vee (X_1x \wedge Q_bx \wedge X_1y) \\ \vee & (X_1x \wedge Q_bx \wedge X_2y) \vee (X_2x \wedge Q_bx \wedge X_3y) \\ \vee & (X_3x \wedge Q_ax \wedge X_3y) \vee (X_3x \wedge Q_bx \wedge X_3y) \end{array}$$

- "Final state is reached from state at x"

$$(X_2x \wedge Q_bx) \vee (X_3x \wedge Q_ax) \vee (X_3x \wedge Q_bx)$$

Putting all of the ideas together, we will get the following sentence describin the set of words accepted by automaton in Figure 2.

$$\begin{array}{ll} \exists X_1 \exists X_2 \exists X_3 & \forall x((X_1x \vee X_2x \vee X_3x) \wedge \\ & \quad \neg(X_1x \wedge X_2x) \wedge \neg(X_1x \wedge X_3x) \wedge \neg(X_2x \wedge X_3x)) \wedge \\ & \forall x(\mathsf{first}(x) \rightarrow X_1x) \wedge \\ & \forall x \forall y(Sxy \rightarrow ((X_1x \wedge Q_ax \wedge X_1y) \vee (X_1x \wedge Q_bx \wedge X_1y) \\ & \qquad \vee (X_1x \wedge Q_bx \wedge X_2y) \vee (X_2x \wedge Q_bx \wedge X_3y) \\ & \qquad \vee (X_3x \wedge Q_ax \wedge X_3y) \vee (X_3x \wedge Q_bx \wedge X_3y))) \\ & \forall x(\mathsf{last}(x) \rightarrow ((X_2x \wedge Q_bx) \vee (X_3x \wedge Q_ax) \vee (X_3x \wedge Q_bx))) \end{array}$$

We are ready to describe the construction of the sentence for a general automaton using the above ideas.

Proof of Theorem 13 direction $\Leftarrow$. Let A be accepted by NFA $M = (Q, \Sigma, q_0, \delta, F)$, where $Q = \{0, 1, 2, \ldots k\}$, and $q_0 = 0$. We will construct a sentence φ_M such that a word w is accepted by M iff w satisfies φ_M.

φ_M will state that there is a successful run on w. The run will be encoded by predicates X_i (for $i \in \{0, \ldots k\}$) which will hold at a position p if $\mathcal{A}$ is in state i at time p in the run. Thus φ_M is

$$\begin{array}{ll} \exists X_0 \exists X_1 \cdots \exists X_k & \forall x((\bigvee_i X_ix) \wedge \bigwedge_{i \neq j}(\neg(X_ix \wedge X_jx))) \\ & \wedge \forall x(\text{first}(x) \rightarrow X_0(x)) \\ & \wedge \forall x \forall y(Sxy \rightarrow \bigvee_{(i,a,j) \in \delta}(X_ix \wedge Q_ax \wedge X_jy)) \\ & \wedge \forall x(\text{last}(x) \rightarrow \bigvee_{j \in F, (i,a,j) \in \delta}(X_ix \wedge Q_ax)) \quad \square \end{array}$$

The sentence φ_M constructed above is linear in the number of states and transitions in the automaton M.

Before proving the other direction of Theorem 13 it is worth observing that the sentence constructed for a regular language has a special structure — it is sequence of existentially quantified relational variables followed by universally quantified first order variables, and then a formula with no quantifiers. Combined with the converse direction that we will prove in Section 2.2, we can conclude that for MSO on word structures there is a normal form. When interpreted over wordds, every sentence in MSO is equivalent to a sentence that has a sequence of existentially quantified relational variables followed by a sequence of universally quantified first order variables.

2.2 Definability implies Regularity

In this section we will prove that if a language A is defined by a sentence φ, then A is regular. We will prove the regularity of A by constructing an automaton that recognizes A. The natural approach to carry out this construction is by structural induction on the sentence φ. Let us walk through how such a proof might proceed by considering a few cases in this inductive proof will look like. Some of the cases would be as follows.

- Suppose $\varphi = \neg\psi$. By induction, we have M such that $\mathbf{L}(M) = [\![\psi]\!]$. Then automaton for φ is the automaton that recognizes $\overline{\mathbf{L}(M)}$.

- Suppose $\varphi = \exists x\psi$. A word w satisfies φ is there is some assignment to x such that w with that assignment satisfies ψ.

As the case for $\exists x\psi$ indicates, we need to consider the construction of automata not just for sentences, but also for formulas. But if we need to encode the satisfaction of a formula, we also need a way to consider assignments to free variables.

We will encode assignments along with the word, by consider a word over an extended alphabet. Consider a word structure $\mathcal{W}$ over alphabet Σ and assignment α. Suppose we are only interested in the values α assigns to $X_1, \ldots X_i$ and $x_1, \ldots x_j$. Such a pair $(\mathcal{W}, \alpha)$ can be encoded by a string w' over the alphabet $\Sigma' = \Sigma \times \{0,1\}^{i+j}$ as follows: the symbol $(a, c_1, c_2, \ldots c_{i+j})$ appears in position p iff

- $Q_a(p)$ holds in $\mathcal{W}$,
- For $k \leq i$, $c_k = 1$ iff $p \in \alpha(X_k)$,
- For $i+1 \leq k \leq i+j$, $c_k = 1$ iff $\alpha(x_{k-i}) = p$

Let us consider an example to illustrate this encoding.

Example 15. Consider the word $w = abacb$ and assignment α such that $\alpha(X) = \{1,3,4\}$ and $\alpha(x) = 2$. Then the pair (w, α) can be encoded by the word

$$\begin{pmatrix} a \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} b \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} a \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} c \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} b \\ 0 \\ 0 \end{pmatrix}$$

The broad ideas behind our construction of the automaton is as follows. Consider a formula $\varphi(X_1, \ldots X_i, x_1, \ldots x_j)$, whose free variables are contained in $\{x_1, \ldots x_j\}$ and whose free relational variables are contained in $\{X_1, \ldots X_i\}$. We will (inductively) construct an automaton M_φ such that for any word w and assignment α,

$$w \models \varphi[\alpha] \qquad \text{iff} \qquad w' \in \mathbf{L}(M_\varphi)$$

where w' is the word encoding of (w, α) (or rather α's restriction to the free variables/relational variables).

Notation. To describe the automaton it will be useful to introduce some notation, so that the construction is easy to understand. The notation will be somewhat informal to keep the presentation simple. The automaton we will construct will be over the alphabet $\Gamma = \Sigma \times \{0,1\}^{i+j}$. We will find it convenient to highlight only some rows of a particular symbol. So for example,

$$\begin{pmatrix} \# \\ 0 \\ 1 \\ * \end{pmatrix} \quad \begin{array}{l} \# \text{ means any symbol in } \Sigma \text{ and } * \text{ means any vector} \\ \text{in } \{0,1\}^{i+j-2} \end{array}$$

Similarly we will use $\begin{pmatrix} \# \\ 0 \\ * \end{pmatrix}$ to highlight only one row.

Proof of Theorem 13 $\Rightarrow$. The construction of the automaton will be inductive on the structure of the formula. The automaton will often be described in terms of a regular expression, and the meaning would be to use Theorem 7 to construct the automaton corresponding to the regular expression. We will also use the closure properties for regular languages discussed in Section 1.2.

Let us begin by considering the base cases.

- $\bot$: Automaton recognizing $\emptyset$.
- $\mathbf{x} = \mathbf{y}$: Automaton for $\{\begin{pmatrix} \# \\ 0 \\ 0 \\ * \end{pmatrix}\}^* \begin{pmatrix} \# \\ 1 \\ 1 \\ * \end{pmatrix} \{\begin{pmatrix} \# \\ 0 \\ 0 \\ * \end{pmatrix}\}^*$, where the rows for x and y have been highlighted in the regular expression.

- $\mathbf{Q_a x}$: Automaton for $\{\binom{\#}{0}_*\}^*\binom{a}{1}_*\{\binom{\#}{0}_*\}^*$, where the highlighted row is that for x.
- $\mathbf{Sxy}$: Automaton for $\{\binom{\#}{0}_*\}^*\binom{\#}{1}_*\{\binom{\#}{0}_*\}^*\binom{\#}{1}_*\{\binom{\#}{0}_*\}^*$, where the first highlighted row is for x and the second one for y.
- $\mathbf{Xy}$: Automaton for $\{\binom{\#}{0}_*, \binom{\#}{1}_*\}^*\binom{\#}{1}_*\{\binom{\#}{0}_*, \binom{\#}{1}_*\}^*$, where the first highlighted row is for X and the second one for y.

Let us now consider the inductive step.

- $\neg\psi$: The automaton M accepts the language $(\Sigma^* \setminus \mathbf{L}(M_\psi)) \cap$ well-formed, where well-formed contains all words with exactly one 1 in rows corresponding to (first-order) variables.
- $\psi_\mathbf{1} \vee \psi_\mathbf{2}$: The automaton M is the one that accepts $\mathbf{L}(M_{\psi_1}) \cup \mathbf{L}(M_{\psi_2})$.
- $\exists \mathbf{x}\psi$: Let h be the homorphism that maps $(a, c_1, \ldots c_n)$ to $(a, c_1, \ldots c_{n-1})$, where the row being removed is the one corresponding to variable x. Then M must accept the language $h(\mathbf{L}(M_\psi))$.
- $\exists \mathbf{X}\psi$: Let h be the homorphism that maps $(a, c_1, \ldots c_n)$ to $(a, c_1, \ldots c_{n-1})$, where the row being removed is the one corresponding to relational variable X. Then M must accept the language $h(\mathbf{L}(M_\psi))$.

□

Let us now analyze the size of the automaton that we constructed. It is useful the recall the size of automata when proving some of the standard closure properties. Recall that

- Automaton for the union of two languages at most doubles the size
- Automaton for the homomorphic image is linear in the size of the original machine; results in a nondeterministic machine
- Automaton for intersection is quadratic in size
- Automaton for the complement language is exponential because of the need to determinize

Observe that each existential quantification results in an NFA, which if complemented gives an exponential size DFA. Thus, the size of automaton is a tower of exponential where the height of the tower depends on the number of alternations of existential quantification and complementation.

Given the size of the resulting automaton, it is appropriate to ask if this is indeed the best wek can do. Unfortunately, it turns out that there is a matching lower bound.

Theorem 16 (Meyer-Stockmeyer)**.** *There is no translation of MSO formulas of size n to automata, where the size of the automaton can be bounded by a function of the form*

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

where k is a constant, independent of n.

Despite Theorem 16, it turns out that in practice the automata resulting from the translation are not too large. The MONA tool is a decision procedure for MSO on words. It minimizes every intermediate DFA that is produced and it seems to scale in practice.

2.3 Decision Problems and Normal Forms

The classical decision problems of satisfiability and validity of MSO on words can be shown to be decidable as a consequence of Theorem 13. Before presenting these observations, let us recall what these problems are.

Satisfiability Given a MSO sentence φ, determine if there is a word w such that $w \models \varphi$.

Validity Given a MSO sentence φ, determine if for every word w, $w \models \varphi$.

Thus, we are considering satisfiability and validity, when restructed to word models.

Theorem 17. *The satisfiability and validity problems for MSO on words are decidable.*

Proof. Theorem 13 shows that for any sentence φ, there is an automaton M_φ such that $\mathbf{L}(M_\varphi) = [\![\varphi]\!]$. Moreover, there is an effective procedure to construct the automaton M_φ, given φ. Thus, determining if φ is satisfiable is the same as checking if $\mathbf{L}(M_\varphi) \neq \emptyset$, and determining validity is the same as checking if $\overline{\mathbf{L}(M_\varphi)} = \emptyset$. Since the emptiness of regular languages is decidable (Theorem 11), the result follows. □

3 Decidable Theories of Arithmetic

Recall that Gödel's incompleteness theorem says that the set of first order sentence that are true in $(\mathbb{N}, 0, 1, +, \cdot)$ is not recursively enumerable. In this section, we will demonstrate that certain theories of the natural numbers become decidable when the signature has fewer relation symbols.

3.1 Weak Monadic second order theory of one successor (WS1S)

The *weak monadic second-order theory of one successor (WS1S)* is the set of all MSO sentences that are true in the structure $(\mathbb{N}, 0, S)$ under the restriction that all set quantifiers range over *finite* sets. Recall that S is the successor relation, i.e., Sxy iff $y = x + 1$.

Example 18. We can define the ordering relation on natural numbers as follows.

$$<= xy = \exists X Xx \wedge Xy \wedge (\forall u Xu \rightarrow (u = y \vee (\exists v Suv \wedge Xv)))$$

Using this auxiliary formula, we could write the following sentence that belongs to WS1S.

$$\forall x \forall y <= xy \vee <= yx$$

On the other hand, the sentence $\exists X \forall x(Xx \rightarrow (\exists y(Sxy \wedge Xy)))$ is not in WS1S, because no finite non-empty set can satisfy the conditions on X.

Theorem 19. *WS1S is decidable.*

Proof. The proof follows from Theorem 13. For any MSO formula ψ over $(0, S)$, we will show that the set of (weak) assignments α under which ψ holds for the natural numbers is *regular* by constructing an MSO formula on words. This requires us to describe an encoding of assignments to words. This encoding will be very similar to the encoding we used in the proof of Theorem 13.

Let $X_1, X_2, \ldots X_i$ be the set of relational free variables in ψ and $x_1, \ldots x_j$ be the set of first order free variables in ψ. Let $\Sigma_n = \{0,1\}^n$, i.e, n-tuples over $\{0,1\}$. Let α be an assignment such that k is the largest natural number with the property that either $k \in \alpha(X_\ell)$ or $\alpha(x_\ell) = k$, for some index ℓ. Notice that such a k exists because each relational variable is interpreted as a finite set. The assignment α is encoded as a string of length k over Σ_{i+j}, where $\mathsf{str}(\alpha) = b_0 b_1 \cdots b_k$, where for any ℓ, $b_\ell = (a_1, a_2, \ldots a_i, a_{i+1}, \ldots a_{i+j}) \in \Sigma_{i+1}$ such that

- for $m \leq i$, $a_m = 1$ iff $\ell \in \alpha(X_m)$, and

- for $i+1 \leq m \leq i_j$, $a_m = 1$ iff $\ell = \alpha(x_{m-i})$

For example, the assignment $\alpha(X_1) = \{1,3\}$ and $\alpha(x_1) = 2$ is represented by the string

$$\mathsf{str}(\alpha) = \binom{0}{0}\binom{1}{0}\binom{0}{1}\binom{1}{0}$$

As in the proof of Theorem 13, one can show by induction on the structure of the formula ψ, that the collection of assignments α (where every relational varibles is assigned a finite set) such that $(\mathbb{N}, 0, S) \models \psi[\alpha]$ is a regular language.

Thus, for any MSO sentence φ, we can associate an *input free* automaton M_φ, with the property that φ is in WS1S iff M_φ has an accepting computation. Decidability then follows from Theorem 11. □

3.2 Presburger Arithmetic

Though the first order theory of $(\mathbb{N}, 0, 1, +\cdot)$ was proved to not recursively enumerable by Gödel, Presburger showed that $\mathrm{Th}(\mathbb{N}, 0, 1, +)$ is decidable. Thus, "linear" properties of natural numbers can be decided. Presburger proved this result by showing that $(\mathbb{N}, 0, 1, +)$ admits quantifier elimination. Here we are going to present a different proof where we demonstrate the connections between $\mathrm{Th}(\mathbb{N}, 0, 1, +)$ and WS1S. This elegant proof is due to Büchi and Elgot.

Theorem 20 (Presburger 1929)**.** *Th*$(\mathbb{N}, 0, 1, +)$ *is decidable.*

Büchi and Elgot's proof is as follows. For every first order formula $\varphi(x_1, x_2, \ldots x_n)$, we will construct an "equivalent" MSO formula $\varphi'(X_1, X_2, \ldots X_n)$. The decidability of Presburger arithmetic will then follow from the decidability of WS1S. Notice that first order sentence has variables $x_1, \ldots x_n$ while its equivalent MSO formula φ' has only set variables $X_1, \ldots X_n$, one corresponding to each first order variable. For the construction to work, we need to find a way to map assignments of values to variables $x_1, x_2, \ldots x_n$ to an assignment that maps finite sets to variables $X_1, \ldots X_n$. This will be done by identifying a bijective mapping from $\mathbb{N}$ to subsets of $\mathbb{N}$ as follows.

Consider the bijective mapping $\mathrm{Set} : \mathbb{N} \to \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ (where $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ is the collection of all finite subsets of $\mathbb{N}$), defined as follows.

- Represent numbers in reverse binary representation; thus the string $b_0 b_2 \ldots b_n$ is the number $\sum_{i=0}^{n} b_i 2^i$. Thus, 25 is 10011.
- As we have seen, a string $b_0 b_1 \ldots b_n$ represents a set $X = \{i \mid b_i = 1\}$. Thus, 10011 is the set $\{0, 3, 4\}$
- So $\mathrm{Set}(\sum_{i=0}^{n} b_i 2^i) = \{i \mid b_i = 1\}$

Using this mapping we are ready to present a proof of Theorem 20.

Proof of Theorem 20. For any first order formula $\varphi(x_1, \ldots x_n)$ (with free variables among $x_1, \ldots x_n$), we will construct an MSO formula $\mathsf{T}(\varphi)(X_1, \ldots X_n)$ (with free variables among $X_1, \ldots X_n$) such that

$$(\mathbb{N}, 0, 1, +) \models \varphi[\alpha] \qquad \text{iff} \qquad (\mathbb{N}, 0, S) \models \mathsf{T}(\varphi)[[X_i \mapsto \mathrm{Set}(\alpha(x_i))]_{i=1}^{n}]$$

$\mathsf{T}(\varphi)$ will be constructed by structural induction.

Let us begin with the base cases.

- $0 = 0$ or $1 = 1$: $\mathsf{T}(\varphi)$ is $0 = 0$. For $0 = 1$, $\mathsf{T}(\varphi)$ is $\neg(0 = 0)$.
- $0 = x_i$: $\mathsf{T}(\varphi)$ is $\forall y \neg X_i y$.
- $1 = x_i$: $\mathsf{T}(\varphi)$ is $\forall y (y = 0 \leftrightarrow X_i y)$
- $x_i = x_j$: $\mathsf{T}(\varphi)$ is $\forall y X_i y \leftrightarrow X_j y$.

- $+(x_i, x_j, x_k)$*:* We write down the addition algorithm, with an auxiliary set C representing the carrys. So $\mathsf{T}(\varphi)$ is

$$\begin{aligned}\exists C(\quad & \forall x(\mathsf{first}(x) \to \neg Cx) \wedge \\ & \forall x \forall y(Sxy \to (Cy \leftrightarrow \mathsf{AtLeast2}(X_i x, X_j x, Cx))) \\ & \forall x(X_k x \leftrightarrow \mathsf{OddNum}(X_i x, X_j x, Cx)))\end{aligned}$$

  where $\mathsf{AtLeast2}(a, b, c)$ says that "at least 2 out of a,b, and c hold", and $\mathsf{OddNum}(a, b, c)$ says that "an odd number among a,b,c hold".

The inductive cases can be handled as follows.

- $\neg\psi$*:* $\mathsf{T}(\varphi)$ is $\neg\mathsf{T}(\psi)$.
- $\psi_1 \vee \psi_2$*:* $\mathsf{T}(\varphi)$ is $\mathsf{T}(\psi_1) \vee \mathsf{T}(\psi_2)$.
- $\exists x\psi$*:* $\mathsf{T}(\varphi)$ is $\exists X\mathsf{T}(\psi)$.

□