Monadic Second Order

Wild and Wolly Idea: Could we possibly think of a word \( W \in \Sigma^* \) or even \( W \in \Sigma^\omega \) as a structure?

And then cook up some logic that allows us to say that "\( W \) satisfies formula \( \varphi \)?

We could then collect all such \( W \) into a language defined by \( \varphi \) and perhaps use automata to recognize such languages. Maybe?

Building a Structure

A priori, \( W \) is just a sequence of symbols:

\[ W = a_1 a_2 \ldots a_n \]

In order to be able to describe properties of such a word, we need to be able to talk about particular positions in the word. Thus, positions range over \( [n] = \{1, 2, \ldots, n\} \).

For example, we want to say "in position 42 there is a letter a."

Or "position \( x \) is less than position \( y \)."

Variables and Atomic Formulae

We will have variables \( x, y, z, \ldots \) that range over positions in a word, integers in the range \([n]\) where \( n \) is the length of the word.

We allow the following basic predicates between variables:

\[ x < y \quad x = y \]

Of course, we can get, say, \( x \geq y \) by Boolean operations.

Most importantly, we write \( a(x) \) for "there is a letter \( a \) in position \( x \)."

First-Order

We allow quantification for position variables.

\[ \exists x \varphi \quad \forall x \varphi \]

For example, the formula

\[ \exists x, y (x < y \land a(x) \land b(y)) \]

intuitively means "somewhere there is an \( a \) and somewhere, to the right of it, there is a \( b \)."

The formula

\[ \forall x, y (a(x) \land b(y) \Rightarrow x < y) \]

intuitively means "all the \( a \)'s come before all the \( b \)'s."
### Monadic Second-Order

We also have second-order variables $X, Y, Z, \ldots$ that range over sets of positions in a word.

$$\exists X \varphi \land X \varphi \land X(z)$$

Sets of positions are all there is; we do not have variables in our language for, say, binary relations on positions (we do not use full SOL).

This system is called **monadic second-order logic** (with less-than), written $\text{MSO}[<]$.

### Less-Than or Successor

In applications, the atomic relation $x < y$ is slightly more useful than $y = x + 1$, but either one would have the same expressiveness.

On the one hand

$$y = x + 1 \iff x < y \land \forall z (x < z \Rightarrow y \leq z)$$

On the other hand write $\text{closed}(X)$ for the formula $\forall z (X(z) \Rightarrow X(z + 1))$. Then

$$x < y \iff x \neq y \land \forall X (X(x) \land \text{closed}(X) \Rightarrow X(y))$$

This is sometimes written as $\text{MSO}[<] = \text{MSO}[+1]$.

### Semantics

We need some notion of validity:

$$w \models \varphi$$

where $w$ is a word and $\varphi$ a sentence in $\text{MSO}[<]$.

We won’t give a formal definition, but the basic idea is simple. Let $|w| = n$:

- the first order variables range over $[n] = \{1, 2, \ldots, n\}$,
- the second-order variables range over $\mathcal{P}([n])$.

The basic predicates $x < y$ and $x = y$ have their obvious meaning. For the $a(x)$ predicate we let

$$a(x) \iff w_x = a$$

If you are worried about how exactly these structures work, here is the idea.

Example

We can hardwire factors. For example, to obtain a factor $abc$ let

$$\varphi \equiv \exists x, y, z (y = x + 1 \land z = y + 1 \land a(x) \land b(y) \land c(z))$$

Then $w \models \varphi$ iff $w \in \Sigma^*abc\Sigma^*$.

Example

Scattered subwords are very similar in this setting:

$$\varphi \equiv \exists x, y, z (x < y < z \land a(x) \land b(y) \land c(z))$$

Then $w \models \varphi$ iff $w \in \Sigma^*a\Sigma^*b\Sigma^*c\Sigma^*$.
The Language of a Sentence

The examples suggest that, for any sentence $\varphi$, we should consider the collection of all words that satisfy $\varphi$:

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

One cannot fail to notice that, in the examples so far, $L(\varphi)$ is always regular. Needless to say, this is no coincidence.

Also note that we have not used the second-order part of our language yet.

Even/Even

Example

Write even($X$) to mean that $X$ has even cardinality and consider

$$\varphi \equiv \exists X (\forall x (a(x) \implies X(x)) \land \exists y (b(y)))$$

Then $w \models \varphi$ iff $w \in a^*b^*$.

Example

Let first($x$) be shorthand for $\forall z (x \leq z)$, and last($x$) shorthand for $\forall z (x \geq z)$. Then

$$\varphi \equiv \exists x,y (\text{first}(x) \land a(x) \land \text{last}(y) \land b(y))$$

Then $w \models \varphi$ iff $w \in a\Sigma^*b$.

Example

Write even($X$) to mean that $X$ has even cardinality and consider

$$\varphi \equiv \exists X (\forall x (a(x) \iff X(x)) \land \text{even}(X))$$

Then $w \models \varphi$ iff the number of $a$’s in $w$ is even.

We’re cheating, of course; we need to show that the predicate even($X$) is definable in our setting. This is tedious but not really hard:

$$\text{even}(X) \iff \exists Y,Z (X = Y \cup Z \land \text{alt}(Y,Z))$$

Here alt($Y,Z$) is supposed to express that the elements of $Y$ and $Z$ strictly alternate as in

$$y_1 < z_1 < y_2 < z_2 < \ldots < y_k < z_k.$$ 

Missing Pieces

X = Y \cup Z \iff \forall u (X(u) \iff Y(u) \oplus Z(u))

alt(Y,Z) \iff \exists y \in Y \text{first}(y,X) \land \exists z \in Z \text{last}(z,X) \land

$$\forall y \in Y \exists z \in Z (y < z \land \forall x (y < x < z \implies \neg X(x))) \land

\forall z \in Z \exists y \in Y (y < z \land \forall x (y < x < z \implies \neg X(x)))$$

where first($y, X$) and last($z, X$) have the obvious meaning.

Exercise

This does not handle the case where $Y$ and $Z$ are empty; fix.

Show that one can check if the number of $a$’s is a multiple of $k$, for any fixed $k$.

The Link

Definition

A language $L$ is MSO[$\prec$] definable (or simply MSO[$\prec$]) if there is some sentence $\varphi$ such that

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

Our examples suggest the following theorem.

Theorem (Buechi 1960, Elgot 1961)

A language is regular if, and only if, it is MSO[$\prec$] definable.

The theorem connects complexity with definability: we can recognize a set of strings in constant space if, and only if, the set can be described by a formula in our logic.

Formula to Regular (Sketch)

Obviously, the proof comes in two parts:

- For every regular language $L$ we need to construct a sentence $\varphi$ such that $L = L(\varphi)$.

- For every sentence $\varphi$ we have to show that the language $L(\varphi)$ is regular.

We should expect part (1) to be harder since there is no good inductive structure to exploit.

Part (2) is really again model checking: we are going to build a finite state machine that accepts words that satisfy $\varphi$; the argument is very similar to the one for automatic structures, see below.
Regular to Formula (Sketch)

We may safely assume that the regular language $L$ is given by a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

For simplicity assume $Q = [n]$ and $q_0 = 1$.

We have to construct a formula $\varphi$ such that $w \models \varphi$ iff $M$ accepts $w$.

Consider a trace of $M$ on input $w$:

$$q_0 \quad w_1 \quad q_1 \quad w_2 \quad q_2 \quad \ldots \quad q_{n-1} \quad w_m \quad q_m.$$ 

Here $m$ can be arbitrarily large.

We can think of states as being associated with the letters of the word as in:

$$w_1 \quad w_2 \quad w_3 \quad \ldots \quad w_m \quad q_0 \quad q_1 \quad q_2 \quad \ldots \quad q_m.$$ 

Thus, position $x = 1, \ldots, m$ in the word is associated with state $\delta(q_0, w_1 \ldots w_x)$.

The Partition

In order to express this in a MSO$[<]$ formula, we partition the set positions $[m]$ into $n = |Q|$ blocks $X_1, X_2, \ldots, X_n$ such that:

$$X_p(x) \iff \delta(q_0, w_1 \ldots w_x) = p.$$ 

Some of these blocks may be empty, but note that the number of blocks is always exactly $n$ (which we can express as a formula).

But given state $p$ in position $x$ we can determine the state in position $x+1$ given $w_{x+1}$ by a table lookup – which table lookup can be hardwired in a formula.

Expressing Transitions

Technically, this is done by a formula

$$\Phi_{p,a} \equiv \forall x \ (X_p(x) \land a(x+1) \Rightarrow X_{\delta(p,a)}(x+1))$$

meaning "if at position $x$ we are in state $p$ and the next letter is an $a$, then the state in position $x+1$ is $\delta(p,a)$.

Note that this is not quite right, we really need a non-existing position $0$ corresponding to state $q_0$.

Exercise

Figure out how to fix this little glitch. Also figure out how to express "the last state is final."

First-Order

It is natural to ask whether the languages defined by the first-order fragment of MSO$[<]$ have some natural characterization.

A language $L \subseteq \Sigma^*$ is star-free if it can be generated from $\emptyset$ and the singletons $\{a\}$, $a \in \Sigma$, using only operations union, concatenation and complement (but not Kleene star).

Note well: $a^*b^*a^*$ is star-free.

Theorem

A language $L \subseteq \Sigma^*$ is FOL$[<]$ definable if, and only if, $L$ is star-free.

Schützenberger’s Theorem

While we’re at it: star-free languages are quite interesting since they admit a purely algebraic characterization in terms of their syntactic semigroups.

A semigroup is aperiodic if it contains only trivial subgroups (the idempotents of the semigroup).

Theorem (Schützenberger 1965)

A regular language is star-free if, and only if, its syntactic semigroup is aperiodic.
Descriptive Complexity Theory

The Büchi/Elgot theorem establishes a connection between a very low complexity class (constant space) and MSO$^+$. In fact, there is the whole area of descriptive complexity that characterizes complexity classes in terms of logic and finite structures:

- NP corresponds to existential SOL (Fagin 1974).
- PH corresponds to SOL.
- PSPACE corresponds to SOL plus a transitive closure operator.

Exercise

Figure out the details of Fagin’s theorem.

Model Checking

Given a sentence $\varphi$ in MSO$<[\cdot]$, it remains to construct an automaton $A_{\varphi}$ that accepts $L(\varphi)$; this will solve in particular the model checking problem for words and MSO$<[\cdot]$.

The argument is admittedly a bit lame for finite words, but becomes really interesting when we consider infinite words.

We are going to construct $A_{\varphi}$ by induction on the build-up of the formula. As usual, one has to confront the problem of free variables.

For example, consider the (silly) formula

$$\varphi \equiv \exists X,x \left( X(x) \land b(x) \right)$$

Adding Tracks

Essentially all we need is an automaton $A_0$ that handles the matrix $X(x) \land b(x)$, then we can project away the tracks for the existential quantifiers.

The trick is to use an automaton that operates not just on the actual string $W$ but also on additional tracks for $x$ and $X$. This is very similar to what we did to build automata for rational relations.

These additional tracks are binary, so the alphabet is $\{a,b\} \times 2 \times 2$.

The Extra Tracks

A binary track corresponding to a first-order variable must contain exactly one 1, which indicates a unique position in $W$. In other words, the track must hold a word in $0^*10^*$.

The binary word in a second order track is arbitrary, think of it as the characteristic function of the set.

Clearly a synchronous automaton reading all the tracks can determine atomic properties $x < y$ and verify that the variable tracks have the right format.

Building Automata

Hence, by structural induction, we only have to worry about Boolean connectives and quantifiers.

Boolean connectives $\land$ and $\lor$ can be handled by building a product automaton or a disjoint union, respectively.

Existential quantifiers are dealt with by projection (erasing a track) and universal quantifiers are reduced to existential ones, plus two negations.

As usual, negation is much harder; it requires determinization and is potentially exponential.

Going Infinite

Wild and Wolly Idea: Does this also make sense for infinite words?

A priori not, there is no way to specify a general word $W \in \Sigma^\omega$ in a finitary manner. But it’s still worth taking a quick look at what happens.

So, our structures are now words

$$W = a_0a_1a_2 \ldots a_n a_{n+1} \ldots$$

and positions correspondingly range over $\mathbb{N}$. 
Adding Tracks

Tracks coding first-order variable are in $0^*10^\omega$.

The binary word in a second order track is arbitrary; again, think of it as the characteristic function of the set.

**Note:** We could force this set to be finite by insisting that the track be in $2^*0^\omega$. This will come in handy in a moment (weak monadic second order logic).

Of course, we would need to develop automata that can check these conditions. If we have the usual closure properties, this would produce a decision algorithm.

---

Towards Infinity

**A Challenge:** Does it make any sense to consider finite state machines on infinite words?

If so, how would this generalization work?

---

Infinite Words

As a matter of principle, infinite words come in two flavors: bi-infinite

$$\Sigma^\omega = \mathbb{Z} \to \Sigma$$

or one-way infinite

$$\Sigma^\omega = \mathbb{N} \to \Sigma$$

Both kinds appear naturally in the analysis of symbolic dynamical systems (reversible and irreversible).

One-way infinite ones can be used to describe the properties of programs that never halt, such as operating systems and user interfaces. Protocols also naturally give rise to infinite descriptions.

---

Adieu Concatenation

Note that neither $\Sigma^\omega$ nor $\Sigma^\omega$ form a semigroup under concatenation in any conceivable sense of the word: there is no way to combine two infinite words by “placing one after the other” and get another infinite word (at least not of the kind that we are interested).

But not that there is an obvious concatenation operation

$$\Sigma^* \times \Sigma^\omega \to \Sigma^\omega$$

and a slightly less obvious one of type

$$\Sigma^\omega \times \Sigma^\omega \to \Sigma^\omega$$

The second one is particularly interesting in conjunction with automata on bi-infinite words, but we won’t go there: the technical details are too messy.

---

No Pattern Matching

The standard acceptance testing problem makes little sense in this setting.

**Problem:** Acceptance

**Instance:** An $\omega$-automaton $A$ and a word $x \in \Sigma^\omega$.

**Question:** Does $A$ accept input $x$?

Presumably $A$ will just be some finite data structure. But there is no general way to specify the input $x$, the space $\Sigma^\omega$ is uncountable. We could consider periodic words or the like, but as stated the decision problem is basically meaningless.

**But:** We might still be able to generalize the logic approach and use these, yet undefined, $\omega$-automata to solve decision problems for appropriate logics.
Automata Recognizing Infinite Words

**Key Question:** How do we modify finite state machines to cope with infinite inputs?

- **Transition system:** same as for ordinary finite state machines.
- **Acceptance condition:** requires work.

What kind of acceptance condition might make sense? For finite words there is a natural answer based on path existence, but for infinite words things become a bit more complicated. Whatever condition we choose, we should not worry about actual acceptance testing, this is a conceptual problem, not an algorithmic one.

\[
\omega \text{-Languages}
\]

So we are interested in one-way infinite words:

\[
\Sigma^\omega = \mathbb{N} \rightarrow \Sigma
\]

One-way infinite words are often called \( \omega \)-words.

Subsets of \( \Sigma^\omega \) are called \( \omega \)-languages.

Given an automaton for infinite words its acceptance language is denoted by \( L(\mathcal{A}) \) and so on.

Note that an \( \omega \)-language may well be uncountable; there cannot be a good notation system for \( \omega \)-words.

We will usually drop the \( \omega \) whenever it is obvious from context.

Runs and Traces

Since we will not change the underlying transition systems, we can lift the definitions of run and trace to the infinite case: a run is an alternating infinite sequence

\[
\pi = p_0, a_1, p_1, a_2, \ldots, p_{m-1}, a_m, p_m, \ldots
\]

The corresponding infinite sequence of symbols is the trace:

\[
\text{lab}(\pi) = a_1, a_2, \ldots, a_{m-1}, a_m, \ldots \in \Sigma^\omega
\]

In general, the number of runs on a particular input is going to be uncountable, but that will not affect us (it is path existence that matters).

Büchi Automata

**Definition**

A Büchi automaton \( B \) is a transition system \( \langle Q, \Sigma, \tau \rangle \) together with an acceptance condition \( I \subseteq Q \) and \( F \subseteq Q \).

\( B \) accepts an infinite word \( x \in \Sigma^\omega \) if there is a run \( \pi \) of \( B \) on \( x \) that starts at \( I \) and such that \( \text{rec}(\pi) \cap F \neq \emptyset \).

The collection of all such words is the acceptance language of \( B \). A language \( L \subseteq \Sigma^\omega \) is recognizable or \( \omega \)-regular if there is some Büchi automaton that accepts it.

So far, this is just a definition. It seems reasonable, but it is absolutely not clear at this point that we will get any mileage out of this.

Example I

Let \( \Sigma = \{a, b\} \) and

\[
L = \{ x \in \{a, b\}^\omega \mid 1 \leq \#_bx < \infty \}
\]

So \( L \) is the language of all words containing at least one, but only finitely many \( b \)'s. This language is recognizable.

In fact, two states suffice. Here is a Büchi automaton for \( L \):

\[
a \rightarrow 1 \\
1 \rightarrow 2 \\
2 \rightarrow a
\]
**Example II**

Let \( \Sigma = \{a, b, c\} \) and

\[
L = \{ x \in \{a, b, c\}^\omega \mid \#_ax = \#_bx = \infty \land \#_cx < \infty \}
\]

So \( L \) contains finitely many \( c \)'s, but infinitely many \( a \)'s and \( b \)'s. This language is also recognizable.

**Example III**

Consider alphabet \( \Sigma = \{a, b, c\} \). Let \( L \) be the language

\( \text{Every } a \text{ is ultimately followed by a } b, \text{ though there may be arbitrarily many } c \text{'s in between, and there may be only finitely many } a \text{'s.} \)

Then \( L \) is recognizable.

**Correctness**

Note that \( 0 \overset{a}{\rightarrow} 1 \) or \( 0 \overset{b}{\rightarrow} 1 \) would also work; it’s not so clear what the canonical automaton looks like.

Correctness proofs are harder than for ordinary automata, they typically involve some (modest) amount of infinite combinatorics. In this case, one might use the following claims. Write \( K \) for the \( \omega \)-language over \( \{a, b\} \) of words containing infinitely many \( a \)'s and \( b \)'s.

- A word \( x \) is in \( L \) iff \( x = uv \) where \( u \in \{a, b, c\}^* \) and \( v \in K \).
- Let \( s \in \{a, b\} \). Then \( sv \in K \) iff \( v \in K \).
- Any infinite path in the SCC \( \{1, 2, 3\} \) touching state 1 infinitely often must use the edges \((2, 3)\) and \((3, 1)\) infinitely often.

**Exercise**

Give a complete proof that the Büchi automaton accepts \( L \).

**Acceptance Testing**

While acceptance testing in general makes no sense, we can still handle the situation when the word is very simple.

**Lemma**

Let \( A \) be a Büchi automaton and \( U = v \star \) an ultimately periodic infinite word. Then it is decidable whether \( A \) accepts \( U \).

In this case, \( U \) has an obvious finite description: the finite words \( v \) and \( u \). A more general case is a computable word: the function \( U : \mathbb{N} \rightarrow \Sigma \) is computable.

**Exercise**

Prove the last lemma.

**Streamlining Büchi Automata**

It is clear that a Büchi automaton may have useless states. In particular, any inaccessible state (in the sense of a classical finite state machine) is clearly useless. But there is more: for example, if a final state belongs to a trivial strongly connected component it is useless: the computation can pass through the state at most once, so we might as well remove it from the set of final states.

**Lemma**

Useless states can be removed from a Büchi automaton in linear time.

**Exercise**

Give a careful definition of what it means for a state in a Büchi automaton to be useless. Then produce a linear time algorithm to eliminate useless states.
Union and Intersection 49

Lemma
Recognizable languages of $\Sigma^\omega$ are closed under union and intersection.

Proof. For union simply use the disjoint sum of the Büchi automata.
For intersection, we use a slightly modified product machine construction. The new state set is
$$Q_1 \times Q_2 \times \{0, 1, 2\}$$
The transitions on $Q_1$ and $Q_2$ are inherited from the two given machines.
On the last component we act as follows:
- Move from 0 to 1 at the next input.
- From 1 move to 2 whenever a state in $F_1$ is encountered.
- Reset from 2 to 0 when a state in $F_2$ is encountered.

Proof, contd. 50
The initial states are of the form
$$I_1 \times I_2 \times \{0\}$$
and the final states are
$$F = Q_1 \times Q_2 \times \{0\}$$
The infinitely many visits to $F$ imply infinitely many visits to $F_1$ and $F_2$, and conversely.
\[\square\]

There is a message here: though the construction is similar to the finite case it is a bit more complicated. You have to stay alert. Also note that we have not dealt with complements.

Exercise
Fill in the details in the last construction.

Rational Languages 51

Another piece of evidence for the usefulness our our definition is that recognizable language on infinite words can be written down as a type of regular expression.

Definition
A language $L \subseteq \Sigma^*$ is rational if it is of the form
$$L = \bigcup_{i \in \omega} U_i V_i^\omega$$
where $U_i, V_i \subseteq \Sigma^*$ are all regular.

Since we already have a notation system for regular languages of finite words it is easy to obtain a notation system for recognizable languages of $\omega$-words: add one operation $^\omega$ with the understanding that this operation can only be used once and on the right hand side.

Proof, contd. 50
The initial states are of the form
$$I_1 \times I_2 \times \{0\}$$
and the final states are
$$F = Q_1 \times Q_2 \times \{0\}$$
The infinitely many visits to $F$ imply infinitely many visits to $F_1$ and $F_2$, and conversely.
\[\square\]

There is a message here: though the construction is similar to the finite case it is a bit more complicated. You have to stay alert. Also note that we have not dealt with complements.

Exercise
Fill in the details in the last construction.

Rational Expressions 52

A basic expression has the form
$$\alpha \beta^\omega$$
where $\alpha$ and $\beta$ are ordinary regular expressions. As we will see shortly, sums of these expressions then produce exactly all the recognizable $\omega$-languages.

For the examples from above we have fairly simple expressions
$$a^*b(a+b)^*a^\omega$$
$$((b+c)^*(c+ac^b))^\omega$$

Equivalence Recognizable and Rational 53

Lemma
An $\omega$-language is recognizable if, and only if, it is rational.

Proof. First assume $A$ is a Büchi automaton accepting some language $L$. For each final state $p$ define two new automata
$$A_p^0 = A(I, p) \quad A_p^1 = A(p, p)$$
and let $U_p = L(A_p^0)$, $V_p = L(A_p^1)$. Then
$$L = \bigcup_{p \in F} U_p V_p^\omega$$
since $\text{rec}(\pi) \cap F \neq \emptyset$ implies that one particular state $p \in F$ must appear infinitely often.

Proof, contd. 54
For the opposite direction it suffices to show that $L = UV^\omega$ is recognizable for any regular languages $U$ and $V \neq \emptyset, \{\varepsilon\}$ since recognizable languages are closed under union.

To this end consider two machines $A_0$ and $A_1$ for $U$ and $V$. Join the final states of $A_0$ to the initial states of $A_1$ by $\varepsilon$-moves, and the final states of $A_1$ to the initial states of $A_0$.

Perform $\varepsilon$-elimination to obtain a plain nondeterministic automaton $A$. Set the initial states of $A$ to the initial states of $A_0$ and the final states to the final states of $A_1$.

The resulting Büchi automaton $A$ accepts $L$. \[\square\]