Notes on Logic in Computer Science

Richard Willie

November 12, 2024

Contents

1 First-Order and Monadic Second-Order Logic on Words

In this chapter, we look into the expressive power of first-order and monadic second-order logic when the class of structures is restricted to words, i.e. *word structures*.

§1.1 Recap: first-order logic

Write the rest of this.

§1.1.1 Syntax and semantics

Write the rest of this.

§1.2 First-order logic on words

Write the rest of this.

§1.2.1 Syntax and semantics

Write the rest of this.

§1.3 Expressive power of FO(Σ)

Write the rest of this.

§1.4 Recap: second-order logic

Second-order logic is an extension of first-order logic. Recall that one of the main features of first-order logic over propositional logic, was the ability to quantify over elements that are in the universe of the structure. Second-order logic not only allows one to quantify over elements of the universe, but in addition, also allows quantifying relations over the universe.

To help illustrate the distinction between first-order and second-order logic, it's helpful to consider an example from number theory. Here, the objects of our study are the natural numbers $0, 1, 2, \ldots$ and their arithmetic. Using first-order logic, we can make statements about these numbers through atomic expressions such as $x = y$, $x + y = z$, and $x \times y = z$. These expressions are combined with logical connectives like $\wedge$, $\neg$, $\vee$, and $\rightarrow$, as well as quantifiers $\forall x$ and $\exists x$, where the variables $x, y, z, \ldots$ range over the natural numbers.

In second-order logic, we can express even more complex ideas. In addition to variables like $x, y, z, \ldots$ that represent numbers, second-order logic introduces variables $X, Y, Z, \ldots$ that represent properties or relations between numbers. For these second-order variables, we also have quantifiers $\forall X$ and $\exists X$. This means we can now quantify over properties or relations, not just individual elements. Moreover, in addition to first-order atomic expressions, we have new atomic expressions of the form $X(y_1, \ldots, y_n)$.

§1.4.1 Syntax and semantics

Like first-order logic, second-order logic is defined over a *signature*.

Definition 1.4.1 (Signatures in second-order logic)
A *signature* is $\tau = \{\mathcal{C}, \mathcal{F}, \mathcal{R}\}$, where

1. $\mathcal{C} = \{c_1, c_2, \ldots\}$ is a set of constant symbols,
2. $\mathcal{F} = \{f_1^1, f_2^1, \ldots, f_1^2, f_2^2, \ldots, f_j^k, \ldots\}$ is a set of function symbols, where the superscript indicates the arity of the function, and
3. $\mathcal{R} = \{R_1^1, R_2^1, \ldots, R_1^2, R_2^2, \ldots, R_j^k, \ldots\}$ is a set of relation symbols, where the superscript indicates the arity of the relation.

Second-order logic has several kinds of *variables*. It has *individual* or *first-order variables* denoted by lower case letters $x, y, z, \ldots$ possibly with subscripts. It has *property* or *relation variables* denoted by uppercase letters $X, Y, X, \ldots$ possibly with subscripts. Finally, it has *function variables* denoted by uppercase letters $F, G, H, \ldots$ possibly with subscripts. Each relation and function variable has an arity, which is a positive natural number. We identify property variables with 1-ary relation variables.

Remark 1.4.1 — It is noteworthy that although we have property variables, we do not have variables for properties of properties. Such variables would be part of the formalism of third-order logic.

Formulas in second-order logic over signature $\tau = (\mathcal{C}, \mathcal{R})$ are sequences of symbols, where each symbol is one of the following.

1. The symbol $\bot$, called *false*.
2. The symbol $=$, called *equality*.
3. An element of the infinite set $\mathcal{V}_1 = \{x_1, x_2, \ldots\}$ of (first-order) *variables*.
4. An element of the infinite set $\mathcal{V}_2 = \{X_1^1, X_2^1, \ldots, X_1^2, X_2^2, \ldots, X_j^k, \ldots\}$ of *relational variables*, where the superscript indicates the arity of the variable.
5. Constant symbols and relation symbols in τ.
6. The symbol $\neg$, called *negation*.
7. The symbol $\vee$, called *disjunction*.
8. The symbol $\exists$, called the *existential quantifier*.
9. The symbols (and), called *parentheses*.

Remark 1.4.2 — For simplicity, we restrict the signature τ in the definition above to include only constant symbols and relation symbols. In other words, we exclude function symbols and function variables as, after all, functions are just special kinds of relations. In addition, we also omit certain logical symbols, such as $\wedge$ (conjunction), $\rightarrow$ (implication), and the universal quantifier $\forall$.

Abuse of Notation 1.4.3. We will, however, use the logical symbols $\wedge$, $\rightarrow$, and $\forall$ when it's convenient. We appeal to the fact that these symbols can be defined in terms of the others.

As always, not all such sequences are formulas; only *well-formed* are formulas in the logic. In order to define well-formed formulas, we first define the set of *terms*.

Definition 1.4.2 (Terms in second-order logic)
A *term* t over signature $\tau = (\mathcal{C}, \mathcal{R})$ is either

1. a constant symbol $c \in \mathcal{C}$, or
2. a variable $x \in \mathcal{V}_1$.

Having defined terms, we can use them to define well-formed formulas.

Definition 1.4.3 (Well-formed second-order formulas)
A *well-formed formula (wff)* over signature $\tau = (\mathcal{C}, \mathcal{R})$ is inductively defined as follows.

1. $\bot$ is a wff.
2. If t_1 and t_2 are terms, then $t_1 = t_2$ is a wff.
3. If $t_1, \ldots, t_k$ are terms and R^k is a relational symbol in $\mathcal{R}$, then $R^k(t_1, \ldots, t_k)$ is a wff.
4. If $t_1, \ldots, t_k$ are terms and X^k is a relational variable in $\mathcal{V}_2$, then $X^k(t_1, \ldots, t_k)$ is a wff.
5. If φ is a wff, then $(\neg\varphi)$ is a wff.
6. If φ_1 and φ_2 are wffs, then $(\varphi_1 \vee \varphi_2)$ are wffs.
7. If φ is a wff and x is a variable in $\mathcal{V}_1$, then $(\exists x \varphi)$ is a wff.
8. If φ is a wff and X^k is a relational variable in $\mathcal{V}_2$, then $(\exists X^k \varphi)$ is a wff.

More succinctly, we could capture the above definitions of terms and well-formed formulas by the following BNF grammar,

$$\begin{aligned} t &::= c \mid x \\ \varphi &::= \bot \mid t = t \mid R(t, \ldots, t) \mid X(t, \ldots, t) \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\exists x \varphi) \mid (\exists X \varphi) \end{aligned}$$

where c is a constant symbol, x is a variable, and X is a relational variable.

Atomic formulas are well-formed formulas that do not have any logical operators, i.e. either of the form

1. $\bot$, or
2. $t_1 = t_2$ where t_1 and t_2 are terms, or
3. $R^k(t_1, \ldots, t_k)$ where R^k is a k-ary relational symbol and $t_1, \ldots, t_k$ are terms, or
4. $X^k(t_1, \ldots, t_k)$ where X^k is a k-ary relational variable and $t_1, \ldots, t_k$ are terms.

Finally, a *literal* is a formula that is either atomic or the negation of an atomic formula.

Remark 1.4.4 — It is interesting to note that in second-order logic we can actually define the identity $t_1 = t_2$ as $\forall X(X(t_1) \leftrightarrow X(t_2))$ and prove the familiar axioms of identity from properties of the implication.

To define the semantics of second-order logic, we first need to establish the *metalanguage* we will use. This requirement has nothing to do with second-order logic but is rather a general feature of semantics due to Tarski [Ta56]. The most common choice for a metalanguage is *set theory*. We thus give a set-theoretical interpretation of second-order logic, where "properties" are understood as sets within a domain, which itself is a set. This set-theoretical interpretation is standard and highlights the key features of second-order logic.

Remark 1.4.5 — It's important to note, however, that not all properties can be represented as sets—for example, the property of being identical to oneself. But when we treat the domain of individual variables as a set, we can meaningfully interpret the properties of individuals within that domain as sets. If we need to interpret second-order logic in a domain too large to be a set, we can instead use the set-theoretical concept of a *class*. (For a discussion on the set/class distinction in the context of second-order logic, see Shapiro [Sh91].)

Like in the case of first-order logic, the semantics of formulas in second-order logic are defined with respect to a *structure* (or equivalently, a *model*). Recall that a structure is a *universe* along with an *interpretation* of the constant symbols and relation symbols in the signature. To define whether a formula holds in a structure, we also need an *assignment*, which interprets free variables. For first-order logic, an assignment was simply a mapping of variables to elements in the universe of the structure. For second-order logic, however, the presence of relational variables, means that an assignment must also give an interpretation of these variables as relations (of the appropriate arity) over the structure. These are formally defined as follows.

Definition 1.4.4 (Structures in second-order logic)
A *τ-structure* $\mathcal{A}$ is a tuple, $(A, \{c^{\mathcal{A}}\}_{c\in\tau}, \{R^{\mathcal{A}}\}_{R\in\tau})$ where

1. A is a non-empty set called the *universe* (or *domain*) of the structure,
2. for each constant symbol $c \in \tau$, $c^{\mathcal{A}} \in U$ is its interpretation, and
3. for each k-ary relational symbol $R \in \tau$, $R^{\mathcal{A}} \subseteq U^k$ is its interpretation.

The structure $\mathcal{A}$ is said to be *finite* if the universe A is finite. The universe of a structure $\mathcal{A}$ will be denoted by $u(\mathcal{A})$.

Definition 1.4.5 (Assignments in second-order logic)
Given a τ-structure $\mathcal{A}$, an *assignment* α over $\mathcal{A}$ is a pair of functions (α_1, α_2) defined as follows.

1. $\alpha_1 : \mathcal{V}_1 \to u(\mathcal{A})$ assigns each individual variable x to a variable $\alpha_1(x) \in u(\mathcal{A})$.
2. $\alpha_2 : \mathcal{V}_2 \to \cup_k (u(A))^k$ assigns each k-ary relational variable X^k to a relation $\alpha_2(X^k) \subseteq (u(A))^k$.

Abuse of Notation 1.4.6. From here on, we will extend α_1 to apply to any term, not just variables. If the term t is a constant symbol c, we take $\alpha_1(t)$ to be $c^{\mathcal{A}}$. For any variable x, we will apply α_1 as per Definition 1.4.5.

We are now ready to define the semantics of a second-order formula in a structure under an assignment.

Definition 1.4.6 (Satisfaction of second-order formulas)
The relation $\mathcal{A}, \alpha \models \varphi$ is inductively defined as follows.

1. $\mathcal{A}, \alpha \not\models \perp$ for all $\mathcal{A}$ and α.
2. $\mathcal{A}, \alpha \models t_1 = t_2$ iff $\alpha_1(t_1) = \alpha_1(t_2)$.
3. $\mathcal{A}, \alpha \models R(t_1, \dots, t_k)$ iff $(\alpha_1(t_1), \dots, \alpha_k(t_k)) \in R^{\mathcal{A}}$.
4. $\mathcal{A}, \alpha \models X^k(t_1, \dots, t_k)$ iff $(\alpha_1(t_1), \dots, \alpha_k(t_k)) \in \alpha_2(X^k)$.
5. $\mathcal{A}, \alpha \models (\neg\varphi)$ iff $\mathcal{A}, \alpha \not\models \varphi$.
6. $\mathcal{A}, \alpha \models (\varphi_1 \vee \varphi_2)$ iff $\mathcal{A}, \alpha \models \varphi_1$ or $\mathcal{A}, \alpha \models \varphi_2$.
7. $\mathcal{A}, \alpha \models (\exists x \varphi)$ iff there exists $a \in u(\mathcal{A})$ such that $\mathcal{A}, \alpha[x \mapsto a] \models \varphi$.
8. $\mathcal{A}, \alpha \models (\exists X^k \varphi)$ iff there exists $S \in (u(\mathcal{A}))^k$ such that $\mathcal{A}, \alpha[X^k \mapsto S] \models \varphi$.

The definition above suggests that the assignment of values to variables (first-order and relational) only matters for the *free* variables, i.e. those that are not quantified. We extend the definition of free and bound occurrences to relational variable as well.

Definition 1.4.7 (Free and bound variables in second-order logic)
In well-formed formulas $(\exists x \varphi_1)$ and $(\exists X^k \varphi_2)$, φ_1 and φ_2 are said to be in scope of the quantifiers $\exists x$ and $\exists X^k$, respectively. Every occurrence of x and X^k in $(\exists x \varphi)$ and $(\exists X^k \varphi)$, respectively, are said to be *bound*. Any occurrences of variables (first-order and relational) that is not bound are said to be *free*.

When comparing different variable assignments and their effect on a given formula φ, those variables that do not occur in the formula or are not free variables in the formula should not affect the fact that a given structure $\mathcal{A}$ logically implies the formula or not. We state this intuitive result in the following.

Lemma 1.4.7 (The relevance lemma for second-order logic)

For a formula φ and assignments α and β that agree on all the free (first-order) variables and free relational variables of φ, we have $\mathcal{A}, \alpha \models \varphi$ iff $\mathcal{A}, \beta \models \varphi$.

Sentences are formulas with no free (first-order) variables or relational variables. Thanks to the relevance lemma, this means that for sentences, the assignment does not determine its satisfaction.

Proposition 1.4.8

For a sentence φ and any two assignments α and β, we have $\mathcal{A}, \alpha \models \varphi$ iff $\mathcal{A}, \beta \models \varphi$. Therefore, we write $\mathcal{A} \models \varphi$ if for any assignment α, $\mathcal{A}, \alpha \models \varphi$.

Satisfiability and validity of second-order formulas are defined in the same way as for first-order logic. In other words, φ is *satisfiable* if there is a structure $\mathcal{A}$ and assignment α such that $\mathcal{A}, \alpha \models \varphi$; and φ is *valid* if for every structure $\mathcal{A}$ and assignment α, $\mathcal{A}, \alpha \models \varphi$.

§1.4.2 Monadic second-order logic

Monadic second-order logic (MSO) is an important fragment of second-order logic. While first-order logic can be seen as the fragment that does not allow relational variables, MSO is the fragment where all relational variables are *monadic*, i.e. of arity 1. Thus, in MSO one can only quantify over sets, as opposed to more general relations.

Abuse of Notation 1.4.9. Since all relational variables are monadic, from now on we will drop the superscript that indicates the variables arity.

Abuse of Notation 1.4.10. Sometimes, we may also skip parentheses for convenience. For instance, instead of writing $R(x)$, we will just write Rx.

Example 1.4.11

Consider the signature of graphs $\tau_G = \{E\}$ which consists of a single binary relation which denotes the edge relation in the graph. We give the following sentence that expresses the fact that the edge relation E encodes a graph that is 3-colorable.

$$\begin{aligned}&\exists X_1 \exists X_2 \exists X_3 ((\forall x (X_1 x \vee X_2 x \vee X_3 x)) \wedge \\ &\qquad\qquad (\forall x \forall y (Exy \rightarrow (\neg(X_1 x \wedge X_1 y) \wedge \neg(X_2 x \wedge X_2 y) \wedge \neg(X_3 x \wedge X_3 y)))))\end{aligned}$$

The sentence above is monadic.

§1.5 Monadic second-order logic on words

In this section, we will study MSO on a restricted class of structures, specifically, *word structures*. Before introducing MSO on words more formally, let us become familiar with it at an intuitive level. We start by fixing an alphabet $\Sigma = \{a, b\}$. MSO on words has three types of atomic formulas:

1. Formulas of the form $Q_a(x)$ or $Q_b(x)$, where x is a variable ranging over the positions of the word. The intended meaning of $Q_a(x)$ if "the letter at position x is a," and the meaning of $Q_b(x)$ is analogous. For instance, the predicate "all letters of the words are as" is expressed by the formula $\forall x\ Q_a(x)$. The language of all words satisfying the formula, called just the language of the formula is $\mathcal{L}(a^*)$.
2. Formulas of the form $x < y$, where x and y range over the positions of the word. The intended meaning is "position x is smaller than (i.e. lies to the left of) position y." For example, the predicate "if some letter is in a, then all subsequent letters are also in as" is expressed by the formula

   $$\forall x \forall y((Q_a(x) \wedge x < y) \rightarrow Q_a(y)).$$

   The language of the formula is $\mathcal{L}(b^*a^*)$. Notice, however, that this is so because $\Sigma = \{a, b\}$. If $\Sigma = \{a, b, c\}$, then the language of the formula is $\mathcal{L}((b + c)^*a^*)$.
3. Formulas of the form $S(x, y)$ where x and y range over the positions of the word. The intended meaning is "y is the next position after x" or "position x plus one is position y." For example, the predicate "the next letter after a is b, and the next letter after b is a" is expressed by the formula

   $$\forall x \forall y(S(x, y) \rightarrow ((Q_a(x) \wedge Q_b(y)) \vee (Q_b(x) \wedge Q_a(y)))).$$

   The language of the formula is $\mathcal{L}((ab)^* + (ba)^*)$.

§1.5.1 Syntax and semantics

Let's start by defining the signature of MSO on words.

Definition 1.5.1 (The signature of MSO(Σ))

Given an alphabet Σ, the signature of MSO(Σ) is fixed to be $\tau_W = (\{Q_a\}_{a\in\Sigma}, <, S)$ where Q_a is a unary relation symbol for every a in the alphabet Σ, and $<$, S are binary relation symbols.

Now, we formally define the syntax of MSO on words, using BNF grammar.

Definition 1.5.2 (The syntax of MSO(Σ))

The set MSO(Σ) of monadic second-order formulas over an alphabet Σ is the set of expressions generated by the following grammar,

$$\varphi ::= Q_a(x) \mid x < y \mid S(x, y) \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\exists x \varphi) \mid (\exists X \varphi)$$

where a is in the alphabet Σ, x, y are variables, and X is a relational variable.

Finally, a *word structure* is defined as follows. Its universe consists of the positions in the word, and the relations Q_a, $<$, and S are interpreted in the "standard" way.

Definition 1.5.3 (Word structures)

Given an alphabet Σ and a word $w \in \Sigma^*$, let n be the length of w. The structure of w is defined by

$$\begin{aligned}\mathcal{W} = (\{1, 2, \ldots, n\},\\ <= \{(1,2),(1,3),\ldots,(1,n),(2,3),\ldots,(2,n),\ldots,(n-1,n)\},\\ S = \{(1,2),(2,3),(3,4),\ldots,(n-1,n)\},\\ \{Q_a = \{x \mid \text{the letter at position } x \text{ is } a\}\}_{a\in\Sigma}).\end{aligned}$$

Write some examples to round out this section.

§1.6 The Büchi-Elgot-Trakhtenbrot theorem

The main result we will establish in this section is that the set of words definable in MSO exactly corresponds to the set of regular languages.

Abuse of Notation 1.6.1. Note that there is a one-to-one correspondence between elements of Σ^* and word structures over Σ. Thus, from now on, we will use w interchangeably to refer both to an element of Σ^* and to the corresponding word structure.

Definition 1.6.1 (MSO-definable languages)

A language A (or, equivalently, a set of words or word structures) over an alphabet Σ is said to be *definable* in MSO if there exists an MSO sentence φ over the signature $\tau_W = (\{Q_a\}_{a\in\Sigma}, <, S)$ such that $A = \{w \mid w \models \varphi\}$.

Theorem 1.6.2 (The Büchi-Elgot-Trakhtenbrot theorem)

A language 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.

Remark 1.6.3 — The Büchi-Elgot-Trakhtenbrot theorem is of fundamental importance in automata theory because it provides a logical characterization of the regular languages. Büchi [Bü60], Elgot [El61], and Trakhtenbrot [Tr62] each independently proved the theorem.

§1.6.1 Regular languages are expressible in MSO(Σ)

In this section, we show that monadic second-order logic on words can express all regular languages. In other words, given any regular language A, we can construct an MSO sentence φ_A that defines the set A.

Lemma 1.6.4 (Regular languages are definable in MSO)

If $A \subseteq \Sigma^*$ is regular, then A is definable in MSO(Σ).

The following presentation of the proof is based on Esparza and Blondin [EsBl23], with some modifications.

Proof. We present a generic procedure that, given a regular language over Σ, represented by a DFA A, constructs a formula φ_A of MSO(Σ) such that $\mathcal{L}(\varphi_A) = \mathcal{L}(A)$. Imagine we are given an A. There is an obvious way to express the membership predicate of $\mathcal{L}(A)$: "a word belongs to $\mathcal{L}(A)$ iff the last state of its run on A is accepting." For this, we introduce the *visit record* of a word. The visit record of a word w is a mapping that assigns to each state q, the set of positions of w after which the run reaches q. It is the inverse of the mapping that assigns to each letter of w the state reached by A after reading it. Formally:

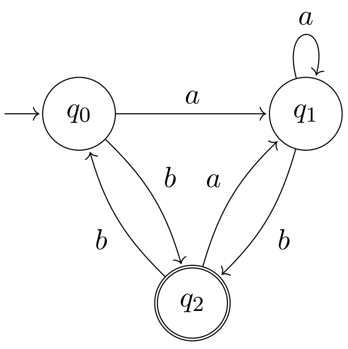
Definition 1.6.2

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and let $w = a_1 \dots a_m$ be a non-empty word over Σ. The visit record of w is the mapping $R_w : Q \to 2^{\{1,\dots,m\}}$ that assigns each state $q \in Q$ to the set of positions defined as follows:

$$R_w(q) = \{i \in \{1, \dots, m\} \mid \hat{\delta}(q_0, a_1 \dots a_i) = q\}.$$

Example 1.6.5

Figure 1.6.1 shows a DFA, its run on the word $w = aabbb$, and the visit record R_w. Observe that each position belongs to the visit record of exactly one state. In other words, $R_w(q_0)$, $R_w(q_1)$, and $R_w(q_2)$ form a partition of the set of positions $\{1, 2, \dots, 5\}$.



Run: $q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_0 \xrightarrow{b} q_2$

Position: 1 2 3 4 5

$$R_w(q_0) = \{4\}$$
$$R_w(q_1) = \{1, 2\}$$
$$R_w(q_2) = \{3, 5\}$$

Figure 1.1: Example of a run and a visit record.

For every non-empty word, "the run of A on the word is accepting" is equivalent to the predicate "the last position of the word belongs to the visit record of an accepting state." Let us examine this predicate. Assume the states of A are $\{q_0, q_1, \dots, q_n\}$, and imagine we are able to define a macro expressing the visit record—that is, a macro VisitRecord$(X_0, \dots, X_n)$ such that an interpretation models the macro if and only if it assigns to each $X_0, \dots, X_n$, the visit record $R_w(q_0), \dots, R_w(q_n)$, respectively. The predicate is then expressed by the sentence

$$\psi_A = \exists X_0 \dots \exists X_n \left(\text{VisitRecord}(X_0, \dots, X_n) \wedge \forall x (\text{last}(x) \to \bigvee_{q_i \in F} X_i(x)) \right)$$

and hence, for every non-empty word w, we have $w \in \mathcal{L}(\psi_A)$ iff $w \in \mathcal{L}(A)$.

The next step is to handle the empty word, which has no visit record. Since the only first-order quantifier of ψ_A is universal, we have $\epsilon \models \psi_A$ for every DFA A, regardless of whether $\epsilon \in \mathcal{L}(A)$ holds or not, and so we cannot define $\varphi_A = \psi_A$. This is easy to fix by defining φ_A as follows:

$$\varphi_A = \begin{cases} \psi_A & \text{if } \epsilon \in \mathcal{L}(A) \\ \psi_A \wedge \exists x(\neg(x < x)) & \text{otherwise} \end{cases}$$

After this adjustment, we have $\mathcal{L}(\varphi_A) = \mathcal{L}(A)$.

To help illustrate, for Example 1.6.5, the corresponding formula is:

$$\exists X_0 \exists X_1 \exists X_2\, (\mathrm{VisitRecord}(X_0, X_1, X_2) \wedge \forall x(\mathrm{last}(x) \rightarrow X_2(x))) \wedge \exists x(\neg(x < x))$$

To complete our proof, we need to construct the macro $\mathrm{VisitRecord}(X_0, \ldots, X_1)$. For this, note that the visit record R_w can also be defined inductively: we first define which component $R_w(q)$ of the record contains position 1, and then, assuming we know which component contains position i, we define which component contains position $i + 1$.

Proposition 1.6.6

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and let $w = a_1 \ldots a_m$ be a non-empty word over Σ. The visit record $R_w : Q \rightarrow 2^{\{1,\ldots,m\}}$ is a unique mapping satisfying the following properties for every $q, q' \in Q$ and every $1 \leq i < m$:

1. $1 \in R_w(q)$ iff $q = \delta(q_0, a_1)$, and
2. if $i \in R_w(q)$ then $i + 1 \in R_w(q')$ where $q' = \delta(q, a_i)$.

Again, to help illustrate, for Example 1.6.5, we get

$$\begin{gathered} 1 \in R_w(q) \text{ iff } q = \delta(q_0, a) = q_1 \\ 2 \in R_w(q) \text{ iff } q = \delta(q_1, a) = q_1 \\ 3 \in R_w(q) \text{ iff } q = \delta(q_1, b) = q_2 \\ \vdots \end{gathered}$$

Intuitively, conditions (1) and (2) of Proposition 1.6.6 state that the initial position belongs to the right component of the visit record and that the visit record "respects" the transition relation of A. We give macros $\mathrm{Init}(X_0, \ldots, X_n)$ and $\mathrm{Respect}(X_0, \ldots, X_n)$ expressing these predicates, where we assume the states of A to be $\{q_0, q_1, \ldots, q_n\}$.

Before we proceed, let us define the following auxiliary macro. Given $0 \leq i \leq n$, we express that position x belongs to X_j iff $i = j$ by

$$\mathrm{InX}_{q_i}(x) = \left(X_i(x) \wedge \bigwedge_{i \neq j} \neg(X_i(x) \wedge X_j(x)) \right).$$

For condition (1), we define

$$\mathrm{Init}(X_0, \ldots, X_n) = \forall x \left(\bigwedge_{a \in \Sigma} (\mathrm{first}(x) \wedge Q_a(x)) \rightarrow \mathrm{InX}_{\delta(q_0,a)}(x) \right).$$

The formula expresses that if the letter at position 1 is an a, the position 1 belongs to $X_{\delta(q_0,a)}$ and to no other set.

Example 1.6.7

For the DFA of Figure 1.6.1 with states $\{q_0, q_1, q_2\}$, we get

$$\begin{aligned}&\mathrm{Init}(X_0, X_1, X_2) = \\ &\qquad \forall x((\mathrm{first}(x) \wedge Q_a(x)) \to \mathrm{InX}_{q_1}(x)) \wedge ((\mathrm{first}(x) \wedge Q_b(x)) \to \mathrm{InX}_{q_2}(x)).\end{aligned}$$

For condition (2), we define

$$\begin{aligned}&\mathrm{Respect}(X_0, \ldots, X_n) \\ &\quad = \forall x \forall y \left(S(x,y) \to \bigwedge_{a\in\Sigma,\ i\in\{0,\ldots,n\}} (Q_a(x) \wedge X_i(x)) \to \mathrm{InX}_{\delta(q_i,a)}(y) \right).\end{aligned}$$

The formula expresses that if a position x belongs to X_i, and the letter at this position is an a, then its successor position y belongs to $X_{\delta(q_i,a)}$, and to no other set.

Example 1.6.8

For the DFA of Figure 1.6.1, this yields

$$\begin{aligned}&\mathrm{Respect}(X_0, \ldots, X_n) \\ &\quad = \forall x \forall y \left(S(x,y) \to \begin{pmatrix} & (Q_a(x) \wedge X_0(x)) \to \mathrm{InX}_{q_1}(y) \\ \wedge & (Q_b(x) \wedge X_0(x)) \to \mathrm{InX}_{q_2}(y) \\ \wedge & (Q_a(x) \wedge X_1(x)) \to \mathrm{InX}_{q_1}(y) \\ \wedge & (Q_b(x) \wedge X_1(x)) \to \mathrm{InX}_{q_2}(y) \\ \wedge & (Q_a(x) \wedge X_2(x)) \to \mathrm{InX}_{q_1}(y) \\ \wedge & (Q_b(x) \wedge X_2(x)) \to \mathrm{InX}_{q_0}(y) \end{pmatrix} \right).\end{aligned}$$

Finally, we are done by setting

$$\mathrm{VisitRecord}(X_0, \ldots, X_n) = \mathrm{Init}(X_0, \ldots, X_n) \wedge \mathrm{Respect}(X_0, \ldots, X_n).$$

□

§1.6.2 Languages expressible in MSO(Σ) are regular

For now, refer to Esparza and Blondin [EsBl23] for a detailed discussion.

Write the rest of this.

Bibliography

[**Bü60**] J Richard Büchi. Weak second-order arithmetic and finite automata. In: *Mathematical Logic Quarterly*, **6**:1-6 (1960) (cited p. 10)

[**El61**] Calvin C Elgot. Decision problems of finite automata design and related arithmetics. In: *Transactions of the American Mathematical Society*, **98**:1 (1961), pp. 21–51 (cited p. 10)

[**EsBl23**] Javier Esparza and Michael Blondin. *Automata theory: An algorithmic approach*. MIT Press, 2023 (cited pp. 11, 13)

[**Sh91**] Stewart Shapiro. *Foundations without foundationalism: A case for second-order logic*. Vol. 17. Clarendon Press, 1991 (cited p. 6)

[**Ta56**] Alfred Tarski. The concept of truth in formalized languages. In: (1956) (cited p. 6)

[**Tr62**] Boris Avraamovich Trakhtenbrot. Finite automata and the logic of one-place predicates. In: *Sibirskii Matematicheskii Zhurnal*, **3**:1 (1962), pp. 103–131 (cited p. 10)