LOGIC AND AUTOMATA ON WORDS AND TERMS

KAMAL LODAYA

1. Words and Languages

Fix a finite set A as an *alphabet*. Its elements are called *letters*. (We will only need to use a, b, c for letters.) A finite sequence of letters $w : \{1, \ldots, n\} \to A$, such as $aabab$, is called a *word* over the alphabet. A set of words is called a *language*.

The set of all words over A, the free monoid formed by the operation of concatenation, is written A^*. The empty word, the identity of the free monoid, is written e. In general, given a language L, L^* is the language formed by concatenating words from L to form another word. For instance, given the language $\{aa, ab, b\}$, also written $aa \cup ab \cup b$, the word $aabab$ is in $\{aa, ab, b\}^*$, but the word ba is not. The empty word e is always in any L^* (by taking words from L zero times).

This notation, with some ad hoc extensions, can be used to describe languages. We will not look into the details of the notation: these are called *regular expressions*, and are defined in any book on automata theory. For example, the expression $\overline{A^*bbA^*}$ describes the complement of A^*bbA^*, that is, the language $NoTwoBs$ of words which do *not* have consecutive occurrences of the letter b.

Could we have a more systematic notation? How about using logical formulas? Here is a formula describing the language $NoTwoBs$:

$$\forall x \forall y (y = x + 1 \wedge b(x) \supset \neg b(y))$$

How about the language $Less = A^*bA^*cA^*$, describing the fact that a b must occur before a c? Here is a formula for it:

$$\exists x \exists y (x < y \wedge b(x) \wedge c(y))$$

Formally, we are working in a structure $\{1, \ldots, n\}$ of *positions* in the word, with variables referring to these positions, some numerical relations, and unary predicate symbols $\{a(.) \mid a \in A\}$. At each position, the predicate corresponding to the letter at that position holds. Each word-structure defines an interpretation for the variables, and a word becomes a model for a sentence (a formula with no free variables). A language, a set of words, becomes a set of *finite models* for the formula.

Exercise. *Write a formula for the language* $S_3 = [((a \cup b)b)^*(a \cup b)ab^*a]^*$.

2. Automata

A little bit of mulling over should convince you that it is pretty difficult to write a formula for the language S_3. Cognitive scientists, like Warren McCulloch and Walter Pitts in the 1940s, invented a graphical notation (called *finite automata*) to represent such languages.

Prepared for the *Summer School on Computability, Logic and Applications*, Calcutta, 24–30 June 1999. I thank the Calcutta Logic Circle for inviting me to speak at the School.

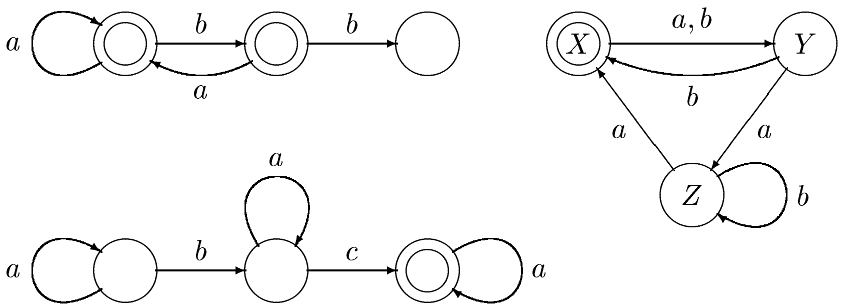

FIGURE 1. Automata for the languages *NoTwoBs*, *Less* and S_3

Formally, a *transition system* is a directed graph (V, E), with the edges (called *transitions*) labelled by letters of the alphabet ($E \subseteq V \times A \times V$). The vertices are called *states*. A finite automaton is a finite transition system with a set of *initial states* and a set of *final states* $I, F \subseteq V$.

The figure shows you automata for the languages we have been talking about. The leftmost state of each automaton is its only initial state, and final states are marked by double circles.

Here is how an automaton operates: It begins in an initial state. On each letter of the word, it takes the corresponding transition from the current state into a (possibly) new state. At the end of the word, if the automaton is in a final state, it *accepts* the word. The *language accepted by the automaton* is all words for which there is a sequence of moves from an initial state to a final state.

Exercise. *Design an automaton which will accept the language Even of all even-length words.*

In 1956, Stephen Kleene proved this fundamental theorem of automata theory:

Theorem 1. *The class of languages defined by regular expressions equals the class of languages accepted by finite automata.*

These languages are called *regular languages.*

Does looking at languages using automata help in our attempt to describe them in logic? Yes, here is a formula for the transition system of the automaton for S_3:

$$
\begin{aligned}
&\exists X \exists Y \exists Z \\
&(\ \forall x \forall y (X(x) \wedge y = x + 1 \supset Y(y)) \\
&\wedge \forall y \forall z (Y(y) \wedge z = x + 1 \supset ((b(y) \supset X(z)) \wedge (a(y) \supset Z(z)))) \\
&\wedge \forall z \forall x (Z(z) \wedge x = z + 1 \supset ((b(z) \supset Z(x)) \wedge (a(z) \supset X(x)))) \\
&)
\end{aligned}
$$

This is a sentence of *(monadic) second-order logic*, with the variables X, Y, Z ranging over unary relations (or sets). Each relation stands for a state of the transition system, and is interpreted as being true for the positions of the word at which the automaton, operating on the word, is in that state. A little more is required to fully describe an automaton —for instance, the initial and final states must be used.

Exercise. *Given an arbitrary finite automaton, make up a formula describing the language accepted by it.*

Formally, our word models are now also equipped with an interpretation for the second-order variables. Think of the automaton alphabet as being expanded with a set of first- and second-order variables, specifying at each position the variables which are true there.

Exercise. *Construct automata on the alphabet* $A \times \wp(\{x, y\}) \times \wp(\{X\})$ *which check if* $a(x)$, $y = x + 1$ *and* $X(y)$ *are true.*

Using these techniques, Richard Büchi and Calvin Elgot in 1960 were able to prove another fundamental result:

Theorem 2. *The class of languages accepted by finite automata equals the class of languages definable in the monadic second-order theory of the successor relation* ($MSO[+1]$)*.*

Proof. The forward direction is answered by an exercise above. For the converse, we use induction on the structure of formulas. Finite automata are constructed for all the atomic formulas (that was another exercise). From automata theory, we know that regular languages are closed under the boolean operations as well as under projection, which we use for the cases dealing with both the first- and second-order quantifiers. □

A little more is true. The formula produced for the forward direction of the proof is an *existential* second-order formula. So going from a second-order formula through an automaton and back amounts to doing an elimination of universal second-order quantifiers. By putting the automata in a standard "minimal" form, we get a "normal form" for an MSO formula. Hence the above theorem yields the following useful corollary. (This was the main motivation for Büchi and Elgot to prove the theorem.)

Corollary 3. *The monadic second-order theory of successor* $MSO[+1]$ *is decidable.*

3. Ehrenfeucht-Fraïssé games

Are second-order formulas really necessary to describe regular languages? This is so, but to show this we have to prove a non-definability result for first-order logic. One way of doing this is to use two-player games devised by Andrzej Ehrenfeucht in the 1960s, based on the work of Roland Fraïssé in the '50s.

Suppose the two players are given two words, *aabaacaa* and *aacaabaa*.

Player I is to distinguish the two words. He picks the position x marked b in the first word.

Player II 's strategy is to identify the two words. She picks the position marked b in the second word.

In the next round, Player I picks the position y of the c in the first word and Player II responds.

Now Player I appeals to the numerical predicate $x < y$. It is true in the first word, but false in the second, so Player I has succeeded in distinguishing the two words and won.

Note the correspondence to the first-order formula $\exists x \exists y(x < y \wedge b(x) \wedge c(y))$.

Exercise. *Suppose the only numerical predicate available is successor (+1). Show that Player II will win in a game with 2 rounds. How many rounds are needed for Player I to win?*

Another game, this time on the words *aaa* and a^n, $n > 3$. Player I picks a first, interior or last position, which Player II has to match. If the game terminates within two rounds, Player II wins. But in 3 rounds, Player I can win.

Finally, a game on the words a^i and a^j. After Round 1, we have one position marked on each word, which we represent as follows:

$$a^i = a^{i_1}\underbrace{a}_{x}a^{i_2},\; a^j = a^{j_1}\underbrace{a}_{x}a^{j_2}$$

Suppose the two indices are equal or both greater than 2. Then Player II will win in the next two rounds. For this to be ensured, at the beginning of the first round, Player II needs $i = j$ or $i, j \geq 7$.

In general, with r rounds ahead, Player II needs to assure that corresponding "delimited" blocks are of the same length or have a length beyond ($\geq$) the *threshold* $2^r - 1$.

In other words, Player II has a *winning strategy* in an r-round game on the two words if $i, j \geq 2^r - 1$. The Ehrenfeucht-Fraïssé theorem relates winning strategies in these games to the quantifier depth of formulas.

Theorem 4. *Player II has a winning strategy in the r-round game on words w_1, w_2 if and only if they satisfy the same formulas of quantifier-depth $\leq r$.*

Proof. By induction on r for both directions. For the base case, Player I can distinguish two words in a zero-round game if and only if they do not satisfy the same atomic formulas. For the induction step from left to right, one more round in the game lifts this "partial isomorphism" by one more pair, which corresponds to formulas with one more quantifier. From right to left, marking a position corresponds to assigning an interpretation to a new first-order variable and now the equivalence upto one higher quantifier-depth means that Player II can still duplicate her opponent's move. A more rigorous proof of this can be found in logic textbooks (for instance, the one by Ebbinghaus and Flum). □

Corollary 5. *The language Even of all even-length words is not definable in $FO[<]$, the first-order theory of a linear order.*

Proof. Suppose *Even* is so definable, by a formula of quantifier depth r. Our work above shows that the words a^{2^r} and a^{2^r-1} cannot be distinguished by such a formula, which is a contradiction. □

This reasoning can be stated more abstractly. A language L is said to be *noncounting* if for some $k > 0$, for all words uvw, $uv^kw \in L$ iff $uv^{k+1}w \in L$. For example, the language a^* is noncounting, but $(aa)^*$ is "counting" —it counts an even number of a's.

Corollary 6. *If a language L is defined by an $FO[<]$ formula, then it is noncounting.*

The converse of this result was shown by Marcel-Paul Schützenberger in 1965. The proof is difficult. Another proof, using the algebraic decomposition of automata (studied by Kenneth Krohn and John Rhodes), is given in the book by Straubing.

What does this mean in terms of automata? It can be shown that, in an automaton in standard form, the existence of a nonempty word v and two distinct states s_1 and s_2 such that the run of a word $uvvw$ reaches s_1 after u, s_2 after uv and then s_1 after uvv makes the language accepted "counting". So definability of a language in $FO[<]$ is decidable.

We have seen that *Even* is accepted by an automaton and therefore defined by an $MSO[+1]$ formula. Since all noncounting languages are regular and definable in $MSO[+1]$, we see that on finite word models, $MSO[+1]$ is strictly more expressive than $FO[<]$.

4. Locality

In the first example of the previous section, we noticed that the words *aabaacaa* and *aacaabaa* could not be distinguished in two rounds using the Ehrenfeucht-Fraïssé game on $FO[+1]$, the first-order theory of successor. This argument can be generalized to show that the language $Less = A^*bA^*cA^*$ cannot be defined using $FO[+1]$. On the other hand, it can be defined in $FO[<]$ using the formula $\exists x \exists y (x < y \wedge b(x) \wedge c(y))$. Since the formula $x < y \wedge \neg \exists z (x < z \wedge z < y)$ defines the predicate $y = x + 1$, $FO[<]$ is strictly more expressive than $FO[+1]$.

In particular, it follows that the $<$ predicate cannot be defined in $FO[+1]$. Another interesting corollary is that the operation of transitive closure cannot be defined in $FO[+1]$ (since $<$ is the transitive closure of successor). On the other hand, $Less$ can be defined in $MSO[+1]$, so adding $<$ to that logic is not necessary.

The weakness of successor is that, in the Ehrenfeucht-Fraïssé game, one cannot relate two positions which can be arbitrarily distant. Given a position x in a structure (word), the isomorphism type of all points a distance $\leq d$ from x is called the *d-sphere* around x. Say a language L is *d, t-testable* if for any two words v and w, and for any d-sphere S, the number of occurrences of S in the two words is either equal or beyond ($\geq$) a threshold t, then $v \in L$ iff $w \in L$. *Locally threshold testable* languages are those that are d, t-testable for some $d, t > 0$.

The argument above was abstracted by W. Hanf and used by Wolfgang Thomas in the 1970s to show the following theorem. The converse is also true.

Theorem 7. *A language definable by an $FO[+1]$ formula is locally threshold testable.*

Exercise. *Try finding an $FO[+1]$ formula (quantifier-depth two is sufficient) for a given locally threshold testable language.*

Once again the quantifier elimination gives a decision procedure for $FO[+1]$ languages.

5. Counting

Clearly, automata can do a restricted amount of counting, where first-order logic by itself is a bit deficient. So what we should do is to consider the theory $FO[N]$, where any arbitrary numerical relation is allowed.

That this idea does not work was shown independently by Miklos Ajtai and by Merrick Furst, James Saxe and Michael Sipser in 1983, who showed that the language $Parity$ of words over a two-letter alphabet, such that one of the letters, say a, occurs an even number of times, cannot be defined in $FO[N]$.

Exercise. *Design an automaton to accept Parity.*

To circumvent this limitation, Howard Straubing, Denis Thérien and Wolfgang Thomas decided to extend first-order logic with generalized quantifiers. Let $n > 1$ and $0 \leq k < n$. The formula $\exists^{k \ mod \ n} x \phi$ is true in a word if the number of positions x where ϕ is true is congruent to k modulo n (that is, $qn + k$ for some integer q). For instance, the language $Even$ is definable by the formula $\exists^{0 \ mod \ 2} x(true)$, and $Parity$ by $\exists^{0 \ mod \ 2} x(a(x))$. Define the logic $FO + MOD$ to have these modulo counting quantifiers for every $n > 0$.

Exercise. *Is the usual existential quantifier ($\exists$) needed once we have the modulo quantifiers?*

Now follow the techniques of the previous section. Say a language L is *d,t-counting mod n* if for any two words v and w, and for any d-sphere S, the number of occurrences of S in the two words is either equal or congruent modulo n beyond ($\geq$) a threshold t, then $v \in L$ iff $w \in L$. *Locally threshold modulo counting* languages are those that are d,t-counting mod n for some $d,t,n > 0$.

Theorem 8. *A language is $FO + MOD[+1]$-definable if and only if it is locally threshold modulo counting.*

There is a quantifier elimination here which gives a decision procedure. In particular, the result shows that the $<$ relation and transitive closure are not definable in $FO + MOD[+1]$.

Just as we defined noncounting languages, we can define *modulo counting* languages and show their equivalence to definability in $FO + MOD[<]$.

After adding all these generalized quantifiers, do we still require second-order logic? Yes. Consider a language for which the automaton in standard form has $s \geq 5$ states. Each word w takes a state in this standard automaton to a (unique) state. That is, w represents a function from $\{1,\ldots,s\}$ to $\{1,\ldots,s\}$. In particular, this function could be a permutation. David Mix Barrington, Kevin Compton, Howard Straubing and Denis Thérien showed that if there are words $w_1, w_2, \ldots, w_{s!}$ which represent every permutation on $\{1,\ldots,s\}$ for this automaton, then the language accepted cannot be defined using $FO + MOD[<]$. (This condition is checkable and therefore yields decidability.)

Exercise. *Design an automaton of the type described above. (Hint: Look at the automaton for S_3.)*

But why should we restrict ourselves to the $<$ predicate? Is there a regular language which cannot be described using $FO + MOD[N]$, where we allow an arbitrary numerical predicate as before? That is an open question. Straubing's book shows how it is related to other open questions in circuit complexity.

6. Terms

The words used in mathematical discourse have more structure than just a string of letters: they are *terms*. The *letters* are function symbols with fixed arities. The set of letters, together with the arity information, is called a *ranked* alphabet or a *signature*. For example, the signature of primary school arithmetic is $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2)$, with $\Sigma_0 = Z$ (integers are the 0-ary constant terms of the language), $\Sigma_1 = \{-\}$ (the unary minus) and $\Sigma_2 = \{+, -, \times, \div\}$.

A term can be thought of as being represented by a Σ-labelled finite tree. For convenience, we will only use proper binary trees (that is, all the interior vertices have two successors). Thus, the term $a + b + c$ is to be thought of as being the tree $((a + b) + c)$. The vertices of the tree are described as a prefix-closed set of words over the alphabet $\{\ell, r\}$. So the empty word e describes the root of the tree. Now a term is a function $t : P \to \Sigma$, with P a prefix-closed subset of $\{\ell, r\}^*$.

The set of all terms over Σ, the free algebra over the signature, is written $\Sigma^\vee$. Note first, that there is no empty term; and second, that if Σ has no constants (0-ary terms), there are no terms either. *Term languages* are subsets of terms.

Seen in this light, the letters of a word alphabet A are unary function symbols, with a 0-ary constant e, and they are written postfix. The term $c(b(a(e))$ is written as the word *abc*. Hence a word is a map to A from a prefix-closed subset P of ℓ^* —which we can (and did) identify with $\{1,\ldots,n\}$. So terms are a genuine generalization of words, as was pointed out by Richard Büchi

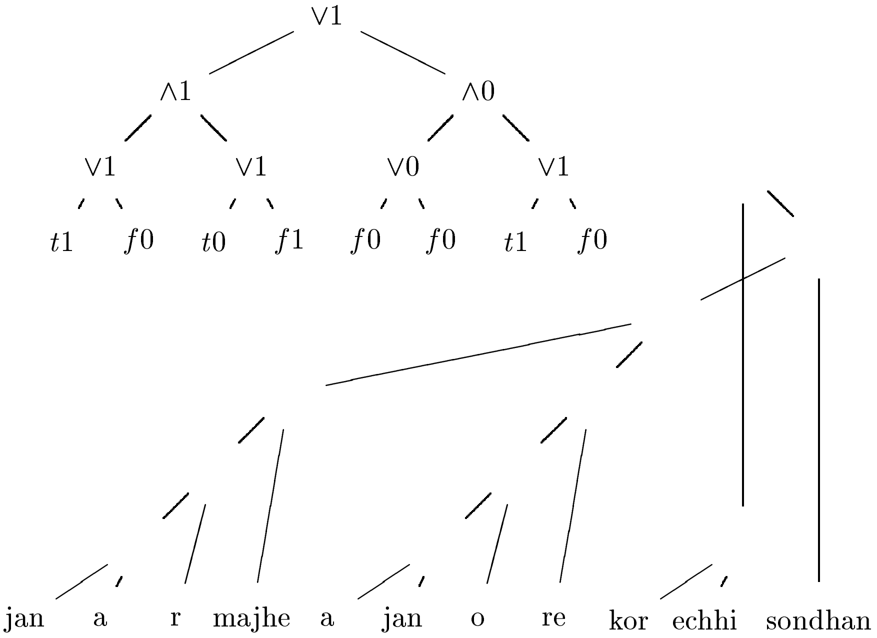

FIGURE 2. Tree automata for the term languages *EvalTrue* and *Chomsky*

and Jesse Wright in the 1960s. (In case you are worried about the reversal, regular languages are closed under reversal, so all the results we saw continue to hold.)

Can we write logical formulas describing term languages? Here is a formula for the language $SumAs = \{a + a + \cdots + a\}$, which, by our convention, is really the term language $\{(\ldots((a + a) + a) + \cdots + a)\}$:

$$\forall x \forall y[[y = xr \supset Leaf(y) \wedge a(y)] \wedge [y = x\ell \supset ((Leaf(y) \supset a(y)) \wedge (\neg Leaf(y) \supset +(y))]]$$

As usual, $+(.)$ is a unary predicate symbol, true at vertices labelled by the $+$ binary operation. The two successor relations on the tree are $y = x\ell$ and $y = xr$, as opposed to the single relation $y = x + 1$ which we had on words. $Leaf(y)$ is an abbreviation for $\neg\exists z(z = y\ell \vee z = yr)$.

Exercise. *Write a formula for the language Anc saying that a* $\times$ *must occur as an ancestor of* $+$.

Exercise. *Let* $SumEvenAs = \{a + a + \cdots + a \mid$ *the number of a's is even*$\}$. *Do you think there is a first-order formula for this language?*

As you might expect, there are *finite tree automata*, which work on term languages. A transition system is now a directed *hypergraph*: for our case, the edges are $E \subseteq (\Sigma_0 \cup (\Sigma_2 \times V \times V)) \times V$. That is, the tree automaton works bottom up: initially, depending upon the constant symbol, it takes a transition from $\Sigma_0 \times V$ and enters a state, and then the state information from two successors is merged using a transition $(\Sigma_2 \times V \times V) \times V$ to reach a "parent" state. Ultimately, the root is reached, and if in a final state, the term is accepted.

The first tree shown in the figure has a signature with constants t, f and binary symbols $\wedge, \vee$. Each vertex of the tree is also labelled with a state 0 or 1 of a tree automaton working on it. As you can see, the tree automaton is evaluating the truth value of the term. If 1 is the only final state, the term language accepted is *EvalTrue*, the set of Boolean terms which evaluate to 1. For the second tree, you are invited to make up your own assignment to states of a tree automaton.

Exercise. *Make up the transition relation for the tree automata described in the figure.*

There is a regular expression syntax for term languages as well, and the fundamental theorems of tree automata theory were proved in the 1960s independently by John Doner and by James Thatcher and Jesse Wright:

Theorem 9. *The class of languages defined by regular expressions, accepted by finite tree automata, and definable in the monadic second-order theory of two successors ($MSO[\ell, r]$), all coincide.*

Corollary 10. *The monadic second-order theory of finitely many successor relations is decidable.*

Proof. First code n-ary trees using binary trees and represent the coding in monadic second-order logic. As in the case of words, the formula for a tree automaton in the theorem above is existential second-order, and we get quantifier elimination and a decision procedure. □

Using Ehrenfeucht-Fraïssé games played on trees, one can show that term languages like $EvalTrue$ and $SumEvenAs$ are not definable in first-order logic. The language $EvalTrue$ is not even definable using modular quantifiers.

There does not seem to be a study of first-order theories with arbitrary tree-definable relations, nor are there any connections to complexity. This area seems to be wide open.

Reading

Howard Straubing: *Finite automata, formal logic and circuit complexity*, Birkhäuser, 1994.
A detailed treatment of the material on word languages in this article. Also gives the connection to circuit complexity.

Wolfgang Thomas: *Languages, automata, and logic*, in *Handbook of formal language theory* **III** (Grzegorz Rozenberg and Arto Salomaa, eds.), Springer, 1997.
Has most of the material of this article. Also goes into more general structures, such as graphs.

More reading

J. Richard Büchi: *Finite automata, their algebras and grammars: Towards a theory of formal expressions* (Dirk Siefkes, ed.), Springer, 1989.
Deals with word as well as tree automata theory. A very readable style — it is interesting to read Büchi's opinions on different aspects of computer science.

Heinz-Dieter Ebbinghaus and Jörg Flum: *Finite model theory*, Springer, 1995.
Deals with EF games, counting quantifiers and a whole lot more, including the connection between logic and more complicated machines.

Yuri Gurevich: *Monadic second-order theories*, in *Model-theoretic logics* (Jon Barwise and Solomon Feferman, eds.), Springer, 1985.
A perspective from the point of view of decidability.

Magnus Steinby: *Tree languages*; Dona Giammarresi and Antonio Restivo: *Two-dimensional languages*; both in *Handbook of formal language theory* **III**, *op. cit.*
The two articles deal with languages over terms and pictures (a generalization over words and terms).

The Institute of Mathematical Sciences, C.I.T. Campus, Chennai 600 113, India.

E-mail address: kamal@imsc.ernet.in