

CDM

Automata and Logic

Klaus Sutner
Carnegie Mellon University

Fall 2011

Outline:

Outline

- 1 The Logic Connection
- 2 Rational Relations
- 3 Deciding Automatic Structures

The Logic Connection:

Approaches to Regularity

So far we have two different approaches to regular languages.

- Constant space: finite state machines.
- Kleene algebra: regular expressions.

But there is more:

- Logic: words as structures.
- Logic II: structures determined by FSMs.

The logic angle leads to some very powerful algorithms that are used e.g. in model checking and formal verification.

The Logic Connection:

Quoi?

The idea is to think of a word as a structure (like a tree, a matrix, the natural numbers, ...) and to use logic to describe the properties of the structure.

This is a bit unusual, but bear with me. First, we need to fix an appropriate language for our logic. As always, we want at least propositional logic.

\perp, \top	constants false, true
\neg	not, negation
\wedge	and, conjunction
\vee	or, disjunction
\rightarrow	implication
\leftrightarrow	equivalence

The Logic Connection:

Variables and Atomic Formulae

We will have variables x, y, z, \dots that range over **positions** in a word, integers in the range 1 through 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

$$Q_a(x)$$

for "there is a letter a in position x ."

The Logic Connection:

Higher Order Variables and Quantifiers

We also have second order variables X, Y, Z, \dots that range over sets of positions in a word. We write $X(x)$ to indicate that $x \in X$.

Sets of positions are all there is; we do not have variables in our language for, say, binary relations on positions. This type of system is often called a monadic logic.

Lastly, we allow quantification for position variables and for sets of positions.

$$\exists x \varphi \quad \forall x \varphi \quad \exists X \varphi \quad \forall X \varphi$$

Definition

This system is called **monadic second order logic** (with less-than), **MSO[<]**.

Semantics

We need some notion of truth:

$$w \models \varphi$$

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

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

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

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

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

Examples

$$aaacbbb \models \forall x (Q_a(x) \vee Q_b(x) \vee Q_c(x))$$

$$aaabbb \models \exists x, y (x < y \wedge Q_a(x) \wedge Q_b(y))$$

$$bbbaaa \not\models \exists x, y (x < y \wedge Q_a(x) \wedge Q_b(y))$$

$$aaabbb \models \exists x, y (x < y \wedge \neg \exists z (x < z \wedge z < y) \wedge Q_a(x) \wedge Q_b(y))$$

$$aaacbbb \not\models \exists x, y (x < y \wedge \neg \exists z (x < z \wedge z < y) \wedge Q_a(x) \wedge Q_b(y))$$

$$aaacbbb \models \exists x (Q_c(x) \wedge \forall y (x < y \rightarrow Q_b(y)))$$

Less-Than or Successor

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

On the one hand

$$y = x + 1 \iff x < y \wedge \forall z (x < z \rightarrow y \leq z)$$

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

Then

$$x < y \iff x \neq y \wedge \forall X (X(x) \wedge \text{closed}(X) \rightarrow X(y))$$

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

Factors and Subwords

Example

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

$$\varphi \equiv \exists x, y, z (y = x + 1 \wedge z = y + 1 \wedge Q_a(x) \wedge Q_b(y) \wedge Q_c(z))$$

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

Example

Subwords are very similar in this setting:

$$\varphi \equiv \exists x, y, z (x < y \wedge y < z \wedge Q_a(x) \wedge Q_b(y) \wedge Q_c(z))$$

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

Some Stars

Example

We can split a word into two parts as in

$$\varphi \equiv \exists x \forall y ((y \leq x \rightarrow Q_a(y)) \wedge (y > x \rightarrow Q_b(y))) \vee \forall x (Q_b(x))$$

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

Example

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

$$\varphi \equiv \exists x, y (\text{first}(x) \wedge Q_a(x) \wedge \text{last}(y) \wedge Q_b(y))$$

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

The Language of a Sentence

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

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

One cannot fail to notice that in the examples so far $\mathcal{L}(\varphi)$ 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 $\text{even}(X)$ to mean that X has even cardinality and consider

$$\varphi \equiv \exists X (\forall x (Q_a(x) \leftrightarrow X(x)) \wedge \text{even}(X))$$

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 $\text{even}(X)$ is definable in our setting. This is tedious but not really hard:

$$\begin{aligned} \text{even}(X) \iff X = \emptyset \vee \\ \exists Y, Z (X = Y \cup Z \wedge \emptyset = Y \cap Z \wedge \text{alt}(Y, Z)) \end{aligned}$$

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

$$y_1 < z_1 < y_2 < z_2 < \dots < y_k < z_k$$

Missing Pieces

$$X = Y \cup Z \iff \forall u (X(u) \leftrightarrow Y(u) \vee Z(u))$$

$$\emptyset = Y \cap Z \iff \neg \exists u (Y(u) \wedge Z(u))$$

$$\text{alt}(Y, Z) \iff \exists y \in Y \forall x < y (\neg Z(x)) \wedge$$

$$\exists z \in Z \forall x > z (\neg Y(x)) \wedge$$

$$\forall y \in Y \exists z \in Z (y < z \wedge \forall x (y < x < z \rightarrow \neg Y(x) \wedge \neg Z(x)))$$

$$\forall z \in Z \exists y \in Y (y < z \wedge \forall x (y < x < z \rightarrow \neg Y(x) \wedge \neg Z(x)))$$

Exercise

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[<] definable** (or simply **MSO[<]**) if there is some sentence φ such that

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

Our examples suggest the following theorem.

Theorem (Buechi 1960)

A language is regular if, and only if, it is MSO[<] 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 φ such that $L = \mathcal{L}(\varphi)$.
- For every sentence φ we have to show that the language $\mathcal{L}(\varphi)$ is regular.

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

Part (2) is by induction on φ but there is the usual technical twist: we need to deal not just with sentences but also with free variables. Since we don't have a formal semantics we will not give details of this construction, but see the next section for a very similar argument.

Regular to Formula (Sketch)

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

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

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

Consider a trace of M on input w :

$$q_0 \ w_1 \ q_1 \ w_2 \ q_2 \ \dots \ q_{m-1} \ w_m \ q_m.$$

Here m can be arbitrarily large.

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

$$\begin{array}{ccccccc} - & w_1 & w_2 & w_3 & \dots & w_m \\ q_0 & q_1 & q_2 & q_3 & \dots & q_m \end{array}$$

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

The Partition

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

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

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

$$\Phi_{p,a} \equiv \forall x (X_p(x) \wedge Q_a(x+1) \rightarrow X_{\delta(p,a)}(x+1))$$

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.”

Expressing Transitions

Now consider the big conjunction of $\Phi_{p,a}$ where $p \in Q$ and $a \in \Sigma$. Add formulae that pin down the first and last state to arrive at a formula of the form

$$\varphi \equiv \exists X_1, \dots, X_n \Psi$$

where Ψ is first order that works as required.

Note that in conjunction with the opposite direction of Büchi’s theorem this result has the surprising consequence that every $\text{MSO}[<]$ formula is equivalent to a $\text{MSO}[<]$ formula containing only one block of existential second order quantifiers.

Exercise

Fill in all the details in the last proof.

Upcoming Attractions

Büchi’s result may seem a bit odd and of no practical values.

However, it also holds for **infinite words** and in that version it is extremely useful.

Variants of Büchi’s theorem are also important in complexity theory.

- The Logic Connection
- ② Rational Relations
- Deciding Automatic Structures

More Logic

We can think of regular languages as “unary relations” on Σ^* . We can manipulate these unary relations nicely since regular languages are closed under lots of operations, in particular the Boolean ones.

Here is a wild and wooly idea: can we generalize to binary relations?

Is there such a thing as a “regular binary relation?”

And, needless to say, what are the closure properties?

One might be tempted to modify finite state machines to work with relations, e.g., by using two input tapes, but there are lots of choices; it is not clear a priori what the right generalization is.

Algebra to the Rescue

We have seen that regular languages can be described algebraically as the closure of \emptyset and singletons $\{a\}$ under the operations union, concatenation and Kleene star, using the monoid Σ^* as the carrier set.

One nice feature of algebra is that a good definition often generalizes: the monoid Σ^* is perhaps the most natural, but there are other plausible choices.

In particular we could use the product monoid $\Sigma^* \times \Sigma^*$: since we are dealing with sets of pairs of strings we naturally obtain binary relations this way.

More Precisely . . .

Suppose we have some monoid M . We are actually working in $\mathfrak{R}(M)$.

- Union comes for free over $\mathfrak{R}(M)$.
- The monoid multiplication naturally carries over to $\mathfrak{R}(M)$.
- Kleene star is defined since $\mathfrak{R}(M)$ is complete.

For $M = \Sigma^*$ we get the usual regular languages this way.

For $M = \Sigma^* \times \Sigma^*$ we get rational relations over alphabet Σ .

Example: Factors

We claim that the relation $\tau(x, y)$ iff x is a factor of y is rational.

To see why, consider the "regular expression"

$$(\varepsilon : a + \varepsilon : b)^* (a : a + b : b)^* (\varepsilon : a + \varepsilon : a)^*$$

This expression defines τ .

Exercise

Show that the relation " x is a subword of y " is also rational.

Rational Relations versus Machines

How does a finite state machine check a binary relation $\rho \subseteq \Sigma^* \times \Sigma^*$? We can use the same nondeterministic machines as before, but the labels will be chosen from

$$\varepsilon : b, a : b, a : \varepsilon,$$

where $a, b \in \Sigma$. The definition of acceptance is the natural one. These machines are often called **transducers** and one speaks of rational relations as **transductions**.

In fact, the labels $a:b$ are strictly speaking redundant, but in practice indispensable.

Theorem

A relation is rational if, and only if, it is recognized by a transducer.

No Boolean Operations

Given two k -ary rational relations ρ and σ on Σ^* it is clear that $\rho \cup \sigma$ is also rational.

Alas, $\rho \cap \sigma$ and $\rho - \sigma$ usually fail to be rational.

As an example, consider

$$R = (a:a)^* (b:\varepsilon)^* \cap (a:\varepsilon)^* (b:a)^*$$

It is easy to see that $R = \{ a^n b^n : a^n \mid n \geq 0 \}$, a non-rational relation.

Exercise

Show that R is indeed non-rational. Show that the relation " x is the reverse of y " is not rational (but note that transductions are closed under reversal).

Relational Composition

Here is a critical, positive result. Recall that for two binary relations ρ and σ on Σ^* their composition is defined to be the binary relation

$$x (\rho \circ \sigma) y \iff \exists z (x \rho z \wedge z \sigma y)$$

Claim

If both ρ and σ are rational then so is their composition $\rho \circ \sigma$.

Proof. Clearly the ternary relation $x \rho z \wedge z \sigma y$ is rational: we can run the machine for ρ on the first two tracks and the machine for σ on the last two.

We can make this product machine "guess" what the proper values for z_i is: just use nondeterminism. Note that one needs to cope with the problem that z may be longer than x and y . \square

The same holds for k -ary ρ and ℓ -ary σ .

Iteration

One might wonder what happens when we move to the transitive reflexive closure ρ^* .

Proposition

The transitive reflexive closure ρ^* of a rational relation is semi-decidable but may be undecidable.

Proof. By definition $\rho^*(x, y)$ iff $\exists k \rho^k(x, y)$, so semi-decidability follows.

To see that ρ^* may be undecidable consider the standard coding of Turing machine configurations as words $u p v \in \Sigma^* Q \Sigma^*$. It is easy to check that the one-step relation for Turing machines is rational. But then $\frac{\perp}{\perp}$, the transitive reflexive closure of this relation, is undecidable in general. \square

Length-Preserving Transductions

How about **length-preserving (lp)** transductions: $x \tau y \Rightarrow |x| = |y|$?

These turn out to be much better behaved. Of course, they are weaker: the one-step relation of a Turing machine is not an lp-transduction (though for a LBA we are OK).

Note that lp-transductions are closed under composition, concatenation and Kleene star.

Theorem

Length-preserving transductions are closed under Boolean operations.

We won't prove these results: one can show that a length-preserving relation is a transduction iff it is automatic as defined in the next section.

Iterating lp-Transductions

Let's only consider the functional case: we are essentially iterating a map $\tau(x) = y$.

Since τ is length-preserving all orbits must be finite, in fact they cannot be longer than $|\Sigma|^{|x|}$.

So, for example, the question "does the orbit of x end in a fixed point?" is trivially decidable.

Proposition

It is undecidable whether all orbits end in a fixed point.

Sketch of proof.

Simulate a Turing machine on a piece of tape of length $|x|$. Set things up so that all orbits end in a fixed point iff the TM diverges on the empty tape.

So the fixed point under τ means: the computation has run out of space. If, on the other hand, the computation converges for some sufficiently long x then we periodically repeat the whole computation. □

Length-Preserving Example

Recall that x^{op} is the word x written backwards.

It is clear that the map $x \mapsto x^{\text{op}}$ cannot be computed by a FSM.

But iteration can be used to "compute" x^{op} as follows.

Define a new alphabet $\Gamma = \Sigma \cup \{\bar{a} \mid a \in \Sigma\}$.

There is a transduction τ such that $\tau(\varepsilon) = \varepsilon$ and

$$\tau(auv) = u\bar{a}v$$

where $au \in \Sigma^+$ and $v \in \Sigma^*$. Let f be the homomorphism $f(\bar{a}) = a$. Then

$$x^{\text{op}} = f(x\tau^* \cap \Sigma^+)$$

■ The Logic Connection

■ Rational Relations

● Deciding Automatic Structures

Machines First

Here is a type of relation that is naturally recognized by a simple class of finite state machines.

Since we are dealing with relations we will use pairs of input words. If only words of the same length are related then we can think of the input as being specified by pairs of letters: to test $x \rho y$ we feed pairs $(x_i, y_i) \in \Sigma \times \Sigma$ to the machine.

Alas, this is not enough, we need to be able to deal with words of different lengths. To this end, define a new alphabet

$$\Gamma = (\Sigma \cup \{\#\}) \times (\Sigma \cup \{\#\})$$

where $\# \notin \Sigma$ is a new **padding symbol**. In essence, $\#$ is the blank symbol on the tape of a Turing machine.

Padding Words

We can think of a word over Γ as split into two tracks, each containing a word over $\Sigma \cup \{\#\}$, where $\#$ occurs only as a suffix.

Given two words $x, y \in \Sigma^*$ we define a word $x:y$ over Γ by padding the shorter word with $\#$'s on the right, if necessary.

$$x:y = \begin{array}{|c|c|c|c|c|c|c|} \hline x_1 & x_2 & \dots & x_n & \# & \dots & \# \\ \hline y_1 & y_2 & \dots & y_n & y_{n+1} & \dots & y_m \\ \hline \end{array}$$

This is sometimes called the **convolution** of x and y .

Note that no padding is needed if x and y have the same length.

Automatic Relations

Here is an idea going back to Büchi and Elgot in 1965.

Definition

A relation $\rho \subseteq \Sigma^* \times \Sigma^*$ is (**synchronously**) **recognizable** or **automatic** if there is finite state machine M over Γ such that

$$\mathcal{L}(M) = \{x:y \mid \rho(x,y)\}$$

k -ary relations are treated similarly.

So, in a sense, recognizable relations are the most computationally simple relations (other than entirely trivial ones such as “always true”).

We are cheating a bit here, these relations are also called *synchronous*.

A cheap example is identity: $x = y$ is an automatic relation.

A moment's thought reveals that $x \neq y$ is also automatic.

Lexicographic Order is Automatic

Given an ordered alphabet Σ consider the binary relation \leq_ℓ on Σ^* defined by

$$x \leq_\ell y \iff x \text{ is a prefix of } y \vee \exists a, b \in \Sigma, u, v, w \in \Sigma^* (x = uav \wedge y = ubw \wedge a < b)$$

Proposition

Lexicographic order is automatic.

Proof. Well ... □

The length-lex order is likewise automatic.

But the relations “ x is a factor of y ” and “ x is a subword of y ” are not automatic (though certainly rational).

Addition is Automatic

Consider the ternary relation α on $\mathbf{2}$ defined by

$$\alpha(x, y, z) \iff \text{bin}(x) + \text{bin}(y) = \text{bin}(z)$$

where $\text{bin}(x)$ is the numerical value of x assuming the LSD is first (reverse binary).

Proposition

Binary addition in reverse binary is automatic.

Proof. The kindergarten algorithm for addition works. □

But note: there is no analogous result for multiplication.

Boolean Operations

Claim

Given two k -ary automatic relations ρ and σ on Σ^ the following relations are also automatic:*

$$\rho \cup \sigma, \rho \cap \sigma, \rho - \sigma.$$

The proof is very similar to the argument for regular languages: one can effectively construct the corresponding automata.

So we can handle disjunctions, conjunctions and negation of automatic relations.

Much better than rational relations, but we need one more ingredient.

Warning: Concatenation and Reversal

Automatic relations are not closed under reversal, though: “ x is a prefix of y ” is automatic, but “ x is a suffix of y ” is not.

Neither are they closed under concatenation (or Kleene star). For example, let

$$R = \{0^i:0^j \mid i, j \geq 0\}$$

$$S = (1:1)^*$$

Then both R and S are automatic, but $R \cdot S$ is not.

Exercise

Prove these assertions about non-closure.

Existential Quantifiers

Here is another important closure property. Suppose ρ is a k -ary relation on Σ^* . Define the **projection** of ρ to be

$$\rho'(x_2, \dots, x_k) \iff \exists z \rho(z, x_2, \dots, x_k)$$

Claim

Whenever ρ is automatic, so is its projection ρ' .

Proof.

Erase the first track in the k -track alphabet:

$$p \xrightarrow{a_1:a_2:\dots:a_k} q \implies p \xrightarrow{a_2:\dots:a_k} q$$

That's it! Of course, the new machine will be nondeterministic in general. □

Automatic Structures

Suppose we have some space whose points are given by words over some alphabet: just think about binary words, representing a fixed number of bits.

Suppose further that the system evolves according to some evolution relation \rightarrow (where $x \rightarrow y$ means that x evolves to y in one step).

So we have a structure

$$\mathcal{C} = \langle \Sigma^*, \rightarrow \rangle$$

where \rightarrow may or may not be deterministic (functional).

Innocent Question: Assuming that \rightarrow is automatic (the structure is **automatic**), what kind of properties of such a system can we check automatically (no pun intended)?

Some Typical Questions

- Is the system reversible?
- Does every state have a predecessor?
- Does every state have exactly k successors?
- Does every state have exactly k predecessors?
- Do all orbits end in a fixed point?
- Is there a limit cycle of length k ?
- Can a set of states B (B as in bad) be reached from some state p ?

A Good Framework

Some of these questions turn out to be undecidable, even if \rightarrow is automatic. But others can be dealt with nicely.

The key trick is to choose the right logic: expressive enough to make interesting assertions, but easy enough so that truth remains decidable.

In this case, plain first order logic is a good first choice. For example

$$\forall y \exists x (x \rightarrow y)$$

means that every state has a predecessor state. And

$$\forall x, y, z (x \rightarrow y \wedge x \rightarrow z \Rightarrow y = z)$$

means that the system is reversible (deterministic).

$$\exists x, y, z (x \rightarrow y \wedge y \rightarrow z \wedge z \rightarrow x \wedge x \neq y)$$

means that there is a 3-cycle somewhere.

Truth for Automatic Structures

So we are given $\mathcal{C} = \langle \Sigma^*, \rightarrow \rangle$: the alphabet is just a table and we have a finite state machine for \rightarrow .

Also, we have a sentence φ in first order logic FOL[\rightarrow] and we would like to know if φ is true over \mathcal{C} .

More technically, we want to solve the following decision problem:

Problem:	FO Validity
Instance:	A automatic structure \mathcal{C} and a FOL sentence φ .
Question:	Is φ valid over \mathcal{C} ?

And, as always, we want good algorithms for this.

The Key Idea: Using Automata

Consider first an atomic formula:

- $\varphi \equiv x_1 = x_2$ or
- $\varphi \equiv x_1 \rightarrow x_2$.

Clearly there is an automaton that checks whether two given words u and v satisfy these formulae. For example we can find \mathcal{A}_\rightarrow such that

$$\mathcal{L}(\mathcal{A}_\rightarrow) = \{ u:v \in \Gamma^* \mid u \rightarrow v \}$$

Incidentally, this is the place where it is useful to assume all structures are purely relational, function symbols are slightly more difficult to deal with.

From Atomic to Quantifier-Free

In the general case suppose we have a formula

$$\varphi(x_1, x_2, \dots, x_k)$$

with all variables as shown.

We construct a k -track machine by induction on the subformulae of φ . The automatic pieces read from two appropriate tracks and check \rightarrow or $=$.

Let's suppose φ is in prenex normal form. The next step is then to construct an automaton for the quantifier-free part φ_0 of φ using standard Boolean operations on the atomic machines from above. We wind up with a machine

$$\mathcal{L}(\mathcal{A}_{\varphi_0}) = \{ u_1:u_2:\dots:u_k \in \Gamma^* \mid \mathcal{C} \models \varphi_0(u_1, u_2, \dots, u_k) \}$$

Dealing with Quantifiers

It is well-known that every FO formula can be written in **prenex normal-form**: all quantifiers are up front, something like

$$\exists x_1 \exists x_2 \forall x_3 \exists x_4 \forall x_5 \varphi(x_1, \dots, x_5)$$

Indeed, there is a simple algorithm to convert an arbitrary formula to PNF.

Since we can eliminate universal quantifiers via $\forall \equiv \neg \exists \neg$ we only need to contend with

$$\exists z \varphi(z, x_1, \dots, x_k)$$

We already know how to do this: Just project $\mathcal{L}(\varphi(z, x_1, \dots, x_k))$: there is an accepting computation in the projected automaton iff there is some appropriate z .

But note that for universal quantifiers this requires determinization to deal with complements.

Finale Furioso

If φ has free variables z_1, \dots, z_ℓ we wind up with an ℓ -track automaton \mathcal{A}_φ that accepts precisely those ℓ -track words that satisfy the formula:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{ u_1 : u_2 : \dots : u_\ell \in \Gamma^* \mid \mathcal{C} \models \varphi(u_1, u_2, \dots, u_\ell) \}$$

If φ is a sentence ($\ell = 0$) we get an unlabeled transition system that has a path from I to F iff the sentence is valid.

Efficiency

- \forall and \exists are linear if we allow nondeterminism.
- \wedge is quadratic via a product machine construction.
- But: \neg is potentially exponential since we need to determinize first. Note that universal quantifiers produce two negations.

So this is a bit disappointing: we may run out of computational steam even when the formula is not terribly large.

A huge amount of work has gone into streamlining this and similar algorithms to deal with instances that are of practical relevance.

3-Cycles

As an example how this might work, consider the question of whether the structure \mathcal{C} contains a 3-cycle.

To keep things simple, suppose that \rightarrow is length-preserving: $x \rightarrow y$ implies $|x| = |y|$.

We need to check

$$\exists x, y, z (x \rightarrow y \wedge y \rightarrow z \wedge z \rightarrow x \wedge x \neq y)$$

So we use the 3-track alphabet $\Gamma = \Sigma \times \Sigma \times \Sigma$.

Given a machine M for \rightarrow we can concoct a machine

$$C_3 = M_{1,2} \times M_{2,3} \times M_{3,1} \times U_{1,2}$$

where $M_{i,j}$ tests if the word in track i evolves to the word in track j .

Machine $U_{i,j}$ tests if the word in track i is different from the word in track j .

Projecting

So we get a machine C_3 that is roughly cubic in the size of the original machine for \rightarrow (disregarding possible savings for accessibility).

Once A is built, we erase all the labels and are left with a digraph.

That digraph has a path from an initial state to a final state if, and only if, there is a 3-cycle in \mathcal{C} .

Note, though, how the machines grow if we want to test for longer cycles: the size of C_k is roughly n^k where n is the size of the original machine – this will not work for long cycles.

Summary

- There is a deep connection between regular languages and certain weak logics, notably monadic second order logic.
- Rational relations can be checked by a finite state machine and are in a sense the simplest non-trivial relations.
- Automatic structures, systems built from automatic relations, have decidable first order properties; the decision algorithm uses automata to express logic.
- Alas, efficiency constraints make the algorithm applicable only in a fairly narrow setting.
- Similar methods are very important in model checking and formal verification.