

Automaticity

KLAUS SUTNER

CARNEGIE MELLON UNIVERSITY

SPRING 2021



1 Properties of Rat

2 Synchronous Relations

3 Model Checking Automatic Structures

Wurtzelbrunft remembers the Banach quote about analogies and immediately concludes:

Every result about regular languages carries over, *mutatis mutandis*, to rational relations.

After all, it's just about the same Kleene algebra we are working in, so what could possibly change? For example, we should be able to come up with a nice machine model, figure out how to determinize and minimize these devices, and so on.

Fortunately, life is so much more interesting than that.

Some results do indeed carry over, almost verbatim. But others are plain false and one has to be very careful not to jump at conclusions.

Consider the binary rational relations

$$A = \left(\begin{smallmatrix} a \\ c \end{smallmatrix}\right)^* \left(\begin{smallmatrix} b \\ \varepsilon \end{smallmatrix}\right)^* \quad B = \left(\begin{smallmatrix} a \\ \varepsilon \end{smallmatrix}\right)^* \left(\begin{smallmatrix} b \\ c \end{smallmatrix}\right)^*$$

Then

$$A \cap B = \left\{ \begin{pmatrix} a^i b^i \\ c^i \end{pmatrix} \mid i \geq 0 \right\}$$

It is easy to see that the intersection cannot be recognized by a finite state transducer, essentially for the same reasons that $\{a^i b^i \mid i \geq 0\}$ fails to be regular.

Exercise

Prove that $A \cap B$ really fails to be rational.

Rational relations are closed under union by definition: we allow nondeterminism.

So the last result shows that we fail to have closure under intersection and complement.

Remember that we ultimately want to tackle first-order logic over simple structures, so this looks like a total fiasco. Indeed, we will have to adjust our definitions in a while.

But for the time being, let's stick with rational relations.

Example

If $K \subseteq \Sigma^*$ and $L \subseteq \Gamma^*$ are regular, then $K \times L$ is rational.

Example

If $\rho \subseteq \Sigma^* \times \Gamma^*$ is rational, then $\text{spt}(\rho) \subseteq \Sigma^*$ and $\text{rng}(\rho) \subseteq \Gamma^*$ are regular.

Example

All the relations “ x is a prefix of y ”, “ x is a suffix of y ”, “ x is a factor of y ” and “ x is a subword of y ” are rational.

Example

Recall the definition of **shuffle**:

$$\varepsilon \parallel y = y \parallel \varepsilon = \{y\}$$
$$xa \parallel yb = (x \parallel yb) a \cup (xa \parallel y) b.$$

So $x \parallel y$ is the set of all possible interleavings of the letters of x and y (preserving relative order). The map $(x, y) \mapsto x \parallel y$ is rational.

Disregarding state complexity, in the world of regular languages, there is no difference between NFAs and DFAs: nondeterminism does not increase the power of the machines.

One might wonder if there is some notion of deterministic rational relation and a corresponding deterministic transducer.

The basic idea is simple: there should be at most one computation on all inputs.

Unfortunately, the technical details are a bit messy (use of endmarkers) and we'll skip this opportunity to inflict mental pain on the student body.

Consider the binary rational relations

$$A = \left(\begin{array}{cc} aa & \\ & b \end{array} \right)^* \quad B = \left(\begin{array}{c} a \\ bb \end{array} \right)^*$$

It is clear that both A and B are deterministic rational relations.

Now consider

$$A \cup B = \left\{ \left(\begin{array}{cc} a^i & \\ & b^j \end{array} \right) \mid i = 2j \vee j = 2i \right\}$$

For the union, your intuition should tell you that nondeterminism is critical: initially, we don't know which type of test to apply. This indicates that determinization is not going to work in general for rational relations (which is to be expected since we already know that complementation fails in general).

Consider the binary relation $<_{\text{len}}$ on Σ^* defined by

$$x <_{\text{len}} y \iff |x| < |y|.$$

We obtain a strict pre-order called **length order**; the corresponding classes of indistinguishable elements are words of the same length.

Given an ordered alphabet Σ consider the binary relation $<_s$ on Σ^* defined by

$$x <_s y \iff \exists a < b \in \Sigma, u, v, w \in \Sigma^* (x = uav \wedge y = ubw)$$

This produces another strict pre-order, the so-called **split order**; this time indistinguishable words are prefixes of one another.

Again assume an ordered alphabet Σ . The **lexicographic order** is a mix of prefix order and split order:

$$x <_{\ell} y \iff x \sqsubset y \vee x <_s y$$

Here $x \sqsubset y$ means that x is a proper prefix of y . Note that lexicographic order is a total order, there are no indistinguishable elements.

Proposition

Length order, split order and lexicographic order are all rational.

Exercise

Construct rational expressions that prove the proposition. Construct transducers that prove the proposition.

Another important way of ordering words is the product order of length order and lexicographic order, the so-called **length-lex order**.

$$x <_{\ell\ell} y \iff x <_{\text{len}} y \vee (|x| = |y| \wedge x <_{\ell} y)$$

Length-lex order is easily seen to be a well-order and there are many algorithms on strings that are naturally defined by induction on length-lex order.

Needless to say, length-lex order is also rational.

Usually one thinks of concatenation as a binary operation. But we can also model it as a ternary relation γ :

$$\gamma(x, y, z) \iff x \cdot y = z$$

Proposition

Concatenation is rational.

Proof. For simplicity assume $\Sigma = \{a, b\}$

$$\gamma = (a:\varepsilon:a + b:\varepsilon:b)^* \cdot (\varepsilon:a:a + \varepsilon:b:b)^*$$

□

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 rational.

Proof. The kindergarten algorithm for addition shows that α is rational. \square

Warning: there is no analogous result for multiplication (for reverse binary encoding; but beware of exotic encodings).

Here is a central result: rational relations are closed under composition. Suppose we have two binary relations $\rho \subseteq \Sigma^* \times \Gamma^*$ and $\sigma \subseteq \Gamma^* \times \Delta^*$. Their composition $\tau = \rho \circ \sigma \subseteq \Sigma^* \times \Delta^*$ is defined to be the binary relation

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

Theorem (Elgot, Mezei 1965)

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

Assume we have transducers \mathcal{A} and \mathcal{B} for ρ and σ , respectively. We may safely assume that the labels in \mathcal{A} have the form a/ε or ε/b where $a \in \Sigma$, $b \in \Gamma$; likewise for \mathcal{B} . Add self-loops labeled ε/ε everywhere.

We construct a product automaton \mathcal{C} with transitions

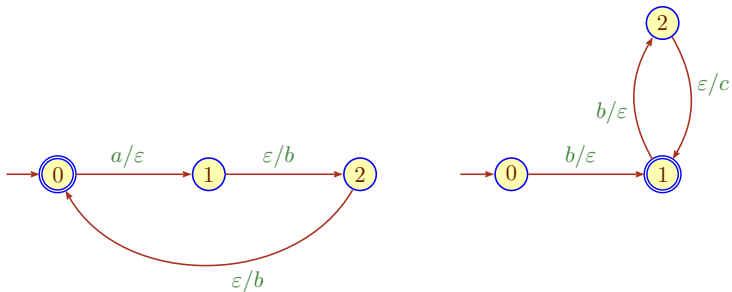
$$(p, q) \xrightarrow{a/c} (p', q')$$

whenever there are transitions $p \xrightarrow{a/b} p'$ and $q \xrightarrow{b/c} q'$ in \mathcal{A} and \mathcal{B} , respectively, for some $a \in \Sigma_\varepsilon$, $b \in \Gamma_\varepsilon$ and $c \in \Delta_\varepsilon$.

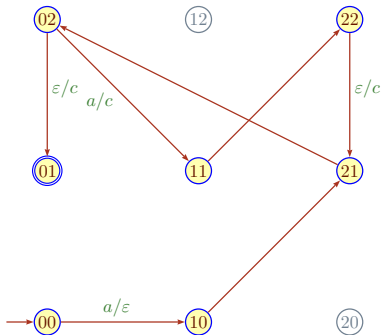
Initial and final states in \mathcal{C} are $I_1 \times I_2$ and $F_1 \times F_2$. It is a labor of love to check that \mathcal{C} accepts x/z if, and only if, $x \rho y$ and $y \sigma z$ for some $y \in \Gamma^*$. \square

Let $\rho = \begin{pmatrix} a \\ bb \end{pmatrix}^*$ and $\sigma = \begin{pmatrix} b \\ \varepsilon \end{pmatrix} \begin{pmatrix} b \\ c \end{pmatrix}^*$; thus $\rho \circ \sigma = \begin{pmatrix} a \\ c \end{pmatrix} \begin{pmatrix} a \\ cc \end{pmatrix}^*$.

Here are the two machines, without the ε/ε self-loops.



And here is the product.



Of course, there is a “better” transducer, but this is the one obtained by blind application of the algorithm.

Here is another important closure property. Suppose ρ is a k -ary relation on words. We define the **projection** of ρ to be

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

Lemma

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

Proof.

Erase the first track in the k -track alphabet:

$$p \xrightarrow{a_1, a_2, \dots, a_k} q \rightsquigarrow p \xrightarrow{a_2, \dots, a_k} q$$

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

One might wonder what happens when we move to the transitive reflexive closure $\text{tcl}(\rho)$. Recall that

$$\text{tcl}(\rho) = \bigsqcup_k \rho^{\circ k}$$

where $\rho^{\circ k}$ indicates the standard iterate, the k -fold composition of ρ with itself.

Mental Health Warning: Unfortunately, the transitive closure is often written ρ^* , in direct clash with the standard notation for the Kleene star of a relation.

Alas, the two are quite incompatible. For example, let ρ be lexicographic order. Clearly, $\text{tcl}(\rho) = \rho$.

But $ab \rho^* aabb$ since $a \rho aa$ and $b \rho bb$. So Kleene star clobbers the order completely.

Theorem

The transitive closure $\text{tcl}(\rho)$ of a rational relation is semidecidable.

Proof.

By definition $x \text{ tcl}(\rho) y$ iff $\exists k (x \rho^{\circ k} y)$.

Obviously, $\rho^{\circ k}$ is primitive recursive, uniformly in k .

So we are conducting an unbounded search over a primitive recursive relation; semidecidability follows. □

What would happen if we add tcl to the closure operations that produce the rational relations?

Theorem

Adding tcl to the closure operations produces precisely all semidecidable relations.

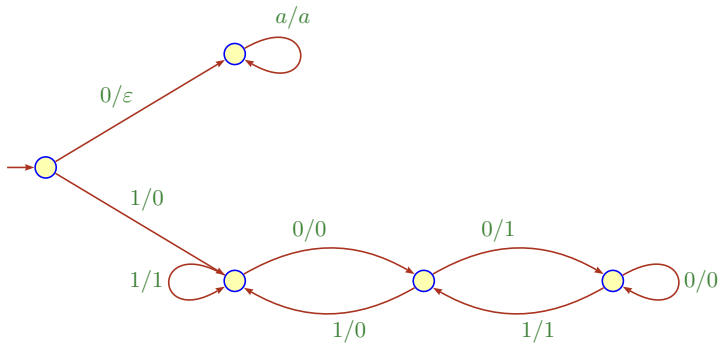
Proof.

Clearly every relation obtained this way is semidecidable.

For the opposite direction, note that the one-step relation of a Turing machine is rational, even synchronous, see below.

Then transitive closure is all that is needed to produce any semidecidable relation.





Recall the infamous Collatz problem: Does the following program halt for all $x \geq 1$?

```
while( x > 1 )           // x positive integer
  if( x even )
    x = x/2;
  else
    x = 3 * x + 1;
```

If we write x in reverse binary, and right-pad with 00, the transducer on the last slide computes one execution of the loop body.

So iterating the composition of the trivial map $x \mapsto x00$ and the transducer leads to an open problem in number theory.

The lower part of the Collatz transducer is a special case of a more general problem: multiply by a fixed constant m . Here we assume the numbers are written in reverse base B .

As always, the key is to pick the right state set. Here is an example: $m = 5$ and $B = 2$.

$$\begin{array}{cccccccc} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & \dots \\ 0 & 0 & a_0 & a_1 & a_2 & a_3 & a_4 & \dots \end{array}$$

If $a_0 = 1$ we need to remember a “carry” from the a_0 in the second row.

It seems reasonable to try $Q = \{0, 1, \dots, m - 1\}$.

We need to extend a partially correct computation by one more step:

$$p_0 \xrightarrow{x/y} p \xrightarrow{a/b} q$$

Note that

$$\text{val}(xa) = \text{val}(x) + aB^{|x|}$$

We may assume

$$m \text{ val}(x) = \text{val}(y) + pB^{|x|}$$

and we need to make sure that

$$m \text{ val}(xa) = \text{val}(yb) + qB^{|xa|}.$$

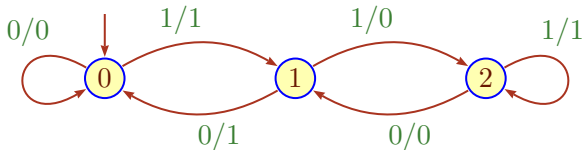
Substituting we get

$$p + am = b + qB$$

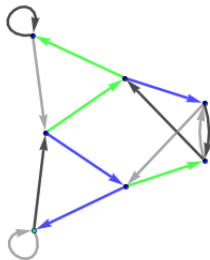
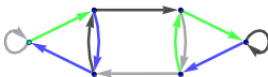
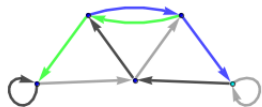
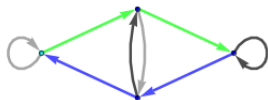
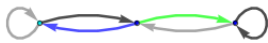
with $0 \leq p, q < m$ and $0 \leq a, b < B$. Letting $s = p + am$ it follows that

$$b = s \bmod B \quad \text{and} \quad q = s \text{ div } B.$$

Here is the “multiply-by-3” transducer for reverse binary.



Note that the input must be padded, otherwise we fail to take the carry into account.



Having to pad the input is clumsy. To get around this, one sometimes augments transducers with an initial and final output maps $\text{inp} : Q \rightarrow \Gamma^*$ and $\text{outp} : Q \rightarrow \Gamma^*$.

The word pair accepted by a run from p to q is then modified from $u \mapsto v$ to

$$u \mapsto \text{inp}(p) v \text{outp}(q)$$

For full-fledged transducers this makes no difference, but it is very convenient for, say, alphabetic transducers.

Life should be much easier with **length-preserving** relations:

$$x \tau y \Rightarrow |x| = |y|$$

In fact, let's only consider the functional case: we are just iterating a map $y = \tau(x)$.

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

Theorem

For length-preserving transductions, transitive closure is PSPACE-complete in general.

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 of a length-preserving transduction 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 length-preserving rational function τ such that $\tau(\varepsilon) = \varepsilon$ and

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

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

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

But as soon as we try to make a global statement, decidability vanishes. For example:

Proposition

It is undecidable whether all orbits of a functional length-preserving transduction end in a fixed point.

Sketch of proof.

Simulate a register machine without input, operating on bounded memory. Set things up so that all orbits end in a fixed point iff the register machine computation diverges.

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

□

In fact this problem turns out to be co-r.e.-complete.

1 **Properties of Rat**

2 **Synchronous Relations**

3 **Model Checking Automatic Structures**

Rational relations in general are just a little too powerful for our purposes, we need to scale back a bit.

One sledge-hammer restriction is to insist that all the relations are **length-preserving**. In this case we have $\rho \subseteq (\Sigma \times \Gamma)^*$, so we are actually dealing with words over the product alphabet $\Sigma \times \Gamma$. These can be checked by an ordinary FSM over this alphabet:

x_1	x_2	\dots	x_n
y_1	y_2	\dots	y_n

Nothing new here, a length-preserving relation is rational iff it is regular as a language over $\Sigma \times \Gamma$.

There is one particularly simple type of transducer that is often useful to recognize length-preserving relations. In a **Mealy machine**, the transitions are described by a function

$$\delta : Q \times \Sigma \longrightarrow \Gamma \times Q.$$

The idea is that transitions are labeled by pairs in $\Sigma \times \Gamma$, so each input letter is transformed into an output letter (alphabetic transducers).

And, the transitions are deterministic.

Exercise

Concoct a binary Mealy machine that implements the successor function modulo 2^k on words of length k (using reverse binary representation).

Alas, length-preserving relations are bit too restricted for our purposes. To deal with words of different lengths, first extend each component alphabet by a **padding symbol** $\#$: $\Sigma_{\#} = \Sigma \cup \{\#\}$ where $\# \notin \Sigma$.

The alphabet for “two-track” words is $\Delta_{\#} = \Sigma_{\#} \times \Gamma_{\#}$.

This pair of padded words is called the **convolution** of x and y and is often written $x:y$.

$$x:y =$$

x_1	x_2	\dots	x_n	$\#$	\dots	$\#$
y_1	y_2	\dots	y_n	y_{n+1}	\dots	y_m

Another example of bad terminology, convolutions usually involve different directions.

Note that we are not using all of $\Delta_{\#}^*$ but only the regular subset coming from convolutions. For example,

a	$\#$	b	$\#$
a	b	a	$\#$

is not allowed.

As always, a similar approach clearly works for k ary relations

$$\Sigma_{1,\#} \times \Sigma_{2,\#} \times \dots \times \Sigma_{k,\#}$$

Exercise

Show that the collection of all convolutions forms a regular language.

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

Definition

A relation $\rho \subseteq \Sigma^* \times \Gamma^*$ is **synchronous** or **automatic** if there is a finite state machine \mathcal{A} over $\Delta_{\#}$ such that

$$\mathcal{L}(\mathcal{A}) = \{x:y \mid \rho(x,y)\} \subseteq \Delta_{\#}^*$$

k -ary relations are treated similarly.

Note that this machine \mathcal{A} is just a language recognizer, not a transducer: since we pad, we can read one symbol in each track at each step.

In a sense, synchronous relations are the most basic examples of relations that are not entirely trivial.

By contrast, one sometimes refers to arbitrary rational relations as asynchronous.

- Lexicographic order is synchronous.
- The prefix-relation is synchronous.
- The ternary addition relation is synchronous.

- The suffix-relation is not synchronous.
- The relations “ x is a factor of y ” and “ x is a subword of y ” are not synchronous.

Our motivation for synchronous relations was taken from length-preserving relations: it is plausible that two words of the same length should be processed in lock-step fashion. The justification for this idea is the following result.

Theorem (Elgot, Mezei 1965)

Any length-preserving rational relation is already synchronous.

The proof is quite messy, we'll skip.

Writing up a nice and elegant proof would be reasonable project.

Claim

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

$$\rho \sqcup \sigma \quad \rho \sqcap \sigma \quad \rho - \sigma$$

The proof is very similar to the argument for regular languages: one can effectively construct the corresponding automata using the standard product machine idea.

This is a hugely important difference between general rational relations and synchronous relations: the latter do form an effective Boolean algebra, but we have already seen that the former are not closed under intersection (nor complement).

Synchronous relations are not closed under concatenation (or Kleene star). For example, let

$$\rho = \begin{pmatrix} a \\ \varepsilon \end{pmatrix}^* \begin{pmatrix} \varepsilon \\ a \end{pmatrix}^*$$
$$\sigma = \begin{pmatrix} b \\ b \end{pmatrix}^*$$

Then both ρ and σ are synchronous, but $\rho \cdot \sigma$ is not (the dot here is concatenation, not composition). Note that $\rho = \begin{pmatrix} a \\ \varepsilon \end{pmatrix}^* + \begin{pmatrix} \varepsilon \\ a \end{pmatrix}^*$ would also work.

Exercise

Prove all examples and counterexamples.

On the upside, synchronous relations are closed under composition.

Suppose we have two binary relations $\rho \subseteq \Sigma^* \times \Gamma^*$ and $\sigma \subseteq \Gamma^* \times \Delta^*$.

Theorem

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

Exercise

Prove the theorem.

More good news: synchronous relations are closed under projections.

Lemma

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

The argument is verbatim the same as for general rational relations: we erase a track in the labels.

Again, this will generally produce a nondeterministic transition system even if we start from a deterministic one. If we also need complementation to deal with logical negation we may have to deal with exponential blow-up.

1 **Properties of Rat**

2 **Synchronous Relations**

3 **Model Checking Automatic Structures**

To simplify matters, suppose we are looking at a structure over the alphabet Σ with just one binary relation:

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

In fact, \rightarrow could be a function, interpreted as a binary relation.

Just to be clear: this scenario includes Turing machines, we could easily code up the one-step relation over a binary alphabet.

So we cannot ask about orbits, only questions that can be phrased in FOL with a single binary predicate (and equality).

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

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

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

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

$$\forall x (\exists y, z ((y \rightarrow x \wedge z \rightarrow x \wedge y \neq z) \wedge \forall u (u \rightarrow x \Rightarrow u = y \vee u = z)))$$

So suppose we have the finite state machines describing $\mathcal{C} = \langle \mathbf{2}^*, \rightarrow \rangle$ and some FO sentence Φ .

As always, we may assume that quantifiers use distinct variables and that the formula is in prenex-normal-form, say:

$$\Phi = \exists x_1 \forall x_2 \forall x_3 \dots \exists x_k \varphi(x_1, \dots, x_k)$$

The matrix $\varphi(x_1, \dots, x_k)$ is quantifier-free, so all we have there is Boolean combinations of atomic formulae.

In our case, there are only two possible atomic cases:

- $x_i = x_j$
- $x_i \rightarrow x_j$

Given an assignment for x_i and x_j (i.e., actual strings) we can easily test these formulae using two synchronous transducers $\mathcal{A}_=$ and $\mathcal{A}_{\rightarrow}$.

So $\varphi(x_1, \dots, x_k)$ defines a k -ary relation over $\mathbf{2}^*$, constructed from \rightarrow and $=$ using Boolean operators. The first step is to build a finite state machine that recognizes this relation.

So the matrix is the quantifier-free formula

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

with all variables as shown. We construct a k -track machine recognizing the corresponding relation by induction on the subformulae of φ .

The atomic pieces read from two appropriate tracks and check \rightarrow or $=$.

More precisely, we use variants $\mathcal{A}_{\rightarrow, i, j}$ and $\mathcal{A}_{=, i, j}$ of the machines from above that check that track i evolves to track j in one step or check equality.

These machines are all defined over the joint alphabet 2^k . Note that the alphabet grows exponentially with k , so there is an efficiency problem.

Suppose $\varphi = \psi_1 \wedge \psi_2$ with corresponding machines \mathcal{A}_{ψ_1} and \mathcal{A}_{ψ_2} . We can use a product machine construction to get \mathcal{A}_{φ} .

Disjunctions are even easier: take the disjoint union.

But negations are potentially expensive: we have to determinize first.

At any rate, we wind up with a composite automaton \mathcal{A}_{φ} that recognizes the relation defined by the matrix:

$$\mathcal{L}(\mathcal{A}_{\varphi}) = \{ u_1:u_2:\dots:u_k \in (\mathbf{2}^k)^* \mid \mathcal{C} \models \varphi(u_1, u_2, \dots, u_k) \}$$

It remains to deal with all the quantifiers in the prefix of Φ . First consider a single existential quantifier, say

$$\exists x \psi(x)$$

We have a machine \mathcal{A}_ψ that has a track for variable x within alphabet 2^ℓ .

Simply erase the x -track from all the transition labels.

This corresponds exactly to existential quantification, done!

For universal quantifiers we use the old equivalence $\forall \equiv \neg \exists \neg$.

This is all permissible, since projections and negations do not disturb automaticity—though they may increase the machine size substantially.

In the process of removing quantifiers, we lose a track at each step and get intermediate machines $\mathcal{B}_{\varphi,\ell}$

$$\mathcal{L}(\mathcal{B}_{\varphi,\ell}) = \{ u_1:u_2:\dots:u_\ell \in \mathbf{2}_\ell^* \mid \mathfrak{C} \models \varphi_\ell(u_1, u_2, \dots, u_\ell) \}$$

for $\ell \leq k$. In the end $\ell = 0$, and we are left with an unlabeled transition system $\mathcal{B}_{\varphi,0}$. This transition system has a path from I to F iff the original sentence Φ is valid.

So the final test is nearly trivial (DFS anyone?), but it does take a bit of work to construct the right machine.

- \forall and \exists are linear if we allow nondeterminism.
- \wedge is at most quadratic via a product machine construction.
- \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.

The question whether there is a k -cycle under \rightarrow requires a bit of care. E.g., the brute-force version of the sentence Φ for $k = 3$ looks like

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

But note that checking for each inequality doubles the size of the machine, so we get something 8 times larger than the machine for the raw 3-cycle. It is much better to realize that since \rightarrow is functional, the last formula is equivalent to

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

Exercise

Figure out how to deal with k -cycles for arbitrary k .

So, based on the better formula, we use the 3-track alphabet $\Sigma = \mathbf{2} \times \mathbf{2} \times \mathbf{2}$ to recognize

$$\{ u:v:w \in \mathbf{2}_3^* \mid u \rightarrow v \rightarrow w \rightarrow u \wedge u \neq v \}$$

Given $\mathcal{A}_{\rightarrow}$, we first concoct a product machine for the conjunctions:

$$\mathcal{C}'_3 = \mathcal{A}_{\rightarrow,1,2} \times \mathcal{A}_{\rightarrow,2,3} \times \mathcal{A}_{\rightarrow,3,1}$$

where $\mathcal{A}_{\rightarrow,i,j}$ tests if the word in track i evolves to the word in track j .

Lastly, we form the product $\mathcal{C}_3 = \mathcal{C}'_3 \times \mathcal{D}_{1,2}$, where $\mathcal{D}_{i,j}$ tests if the word in track i is different from the word in track j . So \mathcal{C}_3 is essentially of two copies of \mathcal{C}'_3 , joined by various transitions.

So we get a machine \mathcal{C}_3 that is roughly cubic in the size of $\mathcal{A}_{\rightarrow}$ (disregarding possible savings for accessibility).

Once \mathcal{C}_3 is built, we erase all the labels and are left with a digraph (since φ has no universal quantifiers there is no problem with negation).

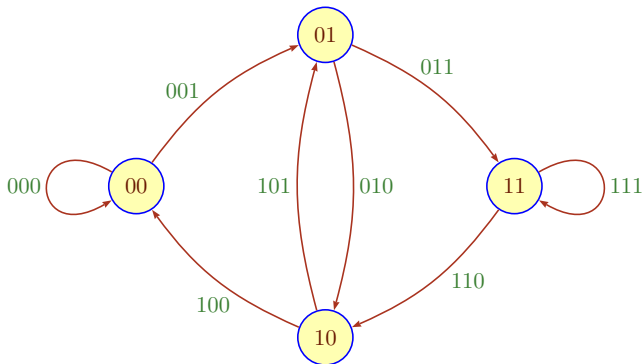
This digraph has a path from an initial state to a final state if, and only if, there is a 3-cycle under \rightarrow .

Note, though, how the machines grow if we want to test for longer cycles: the size of \mathcal{C}_k is bounded only by m^k , where m is the size of $\mathcal{A}_{\rightarrow}$, so this will not work for long cycles. And, we need several products with $\mathcal{D}_{i,j}$.

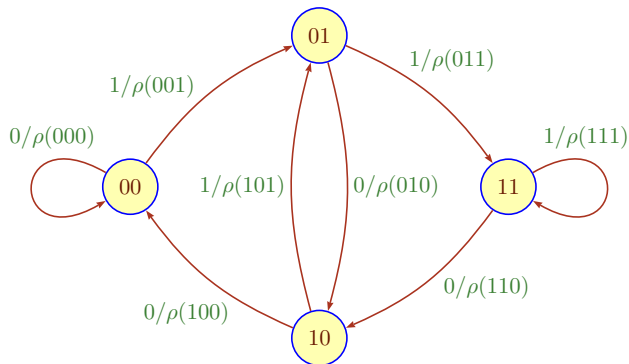
What would the automaton $\mathcal{A}_{\rightarrow}$ for an elementary cellular automaton look like?

This turns out to be a little easier if we consider configurations over $\mathbf{2}^{\mathbb{Z}}$ rather than finite words.

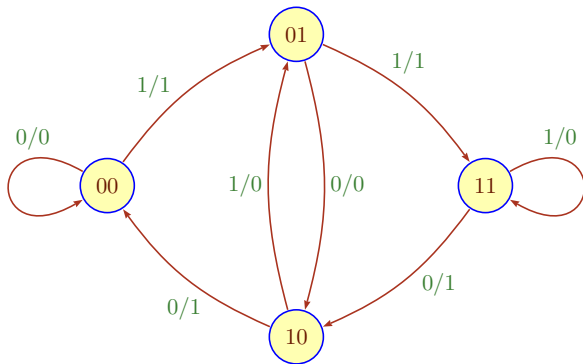
First, an automaton that corresponds to reading a block of length 3, sliding across a configuration.



Each configuration corresponds to exactly one biinfinite path.



If we replace the edge labels xyz by $xyz/\rho(xyz)$, where ρ is the local rule, we get a transducer that corresponds to the global map. All states are initial and final, we are interested in biinfinite runs.



A rather civilized transducer: both the input and output automaton are deterministic.

Recall that defining the global map G_f of a cellular automaton on bitvectors of fixed size n requires to deal with the endpoints: a priori they have no left/right neighbors.

- Cyclic boundary conditions: assume the bitvectors wraps around.
- Fixed boundary conditions: assume there are two phantom bits 0 pre/appended.

So for $x = x_1x_2 \dots x_n$ with FBC we work on $0x_1x_2 \dots x_n0$ leading to n many 3-blocks

$$0x_1x_2, x_1x_2x_3, \dots, x_{n-1}x_n0$$

that we apply the local map f to.

The last transducer works on $2^{\mathbb{Z}}$, but how about plain 2^n ? Say, under fixed boundary conditions? Here is the central problem. We are scanning two words

$$u:v = \begin{array}{|c|c|c|c|} \hline u_1 & u_2 & \dots & u_n \\ \hline v_1 & v_2 & \dots & v_n \\ \hline \end{array}$$

But a synchronous transducer must read the letters in pairs, both read heads move in lockstep.

We need to check whether $v_1 = \rho(0, u_1, u_2)$, and we do not know u_2 after scanning just the first bit pair.

It seems that some kind of look-ahead is required (**memory** versus **anticipation**), but synchronous automata don't do look-ahead, they live in the here-and-now. Looks like we are sunk.

If we drop the synchronicity condition, there is no problem: it is easy to see that \rightarrow is rational. And \rightarrow is clearly length-preserving.

But remember the theorem by Elgot and Mezei:

Rational and length-preserving implies synchronous.

So our relation must be synchronous. Of course, that's not enough: we need to be able to construct the right transducer, not wax poetically about its existence.

Exercise

Show that \rightarrow is rational.

Nondeterminism saves the day: we can guess what x_2 is and then verify in the next step.

Automaton $\mathcal{A}_{\rightarrow}$ uses state set $Q = \{\perp, \top\} \cup \mathbf{2}^3$.

\perp is the initial state, \top the final state and the transitions are given by

$$\begin{aligned} \perp &\xrightarrow{a/e} 0ab & e = \rho(0, a, b) \\ abc &\xrightarrow{c/e} bcd & e = \rho(b, c, d) \\ abc &\xrightarrow{c/e} \top & e = \rho(b, c, 0) \end{aligned}$$

So, this is more complicated than the plain de Bruijn transducer for $\mathbf{2}^{\mathbb{Z}}$.

input	state	condition
—	\perp	—
u_1/v_1	$0 u_1 u_2$	$v_1 = \rho(0 u_1 u_2)$
u_2/v_2	$u_1 u_2 u_3$	$v_2 = \rho(u_1 u_2 u_3)$
u_3/v_3	$u_2 u_3 u_4$	$v_3 = \rho(u_3 u_3 u_4)$
	\vdots	
u_{n-1}/v_{n-1}	$u_{n-2} u_{n-1} u_n$	$v_n = \rho(u_{n-1} u_n 0)$
u_n/v_n	\top	—

A successful computation on input $(u_1 u_2 \dots u_n, v_1 v_2 \dots v_n)$.

Again, the machines produced by our algorithm tend to be very large, so one has to be careful to deal with state-complexity.

One way of getting smaller machines is to rewrite the formula under consideration by hand as in the 3-cycle example. A logically equivalent formula may produce significantly smaller machines. Unfortunately, it can be quite difficult to find better ways to express a first-order property.

If the outermost block of quantifiers is universal, the last check can be more naturally phrased in terms of Universality rather than Emptiness. In this case one should try to use Universality testing algorithms without complementation.

We can easily augment our decision machinery by using additional predicates so long as these predicates are themselves synchronous.

This can be useful as a shortcut: instead of having a formula that defines some property (which then is translated into an automaton), we just build the automaton directly and in an optimal way.

Interestingly, this trick can also work for properties that are not even definable in FOL. We can extend the expressibility of our language and get smaller machines for the logic part that way.