CDM

Automaticity

Klaus Sutner

Carnegie Mellon University

Spring 2023

Wurtzelbrunft's Conjecture

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.

Problem 1: Intersection

Consider the binary rational relations

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

Then

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

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.

Disaster Strikes

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

Example

If $\rho \subseteq \Sigma^\star \times \Gamma^\star$ is rational, then spt$(\rho) \subseteq \Sigma^\star$ and rng$(\rho) \subseteq \Gamma^\star$ 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:

$$\begin{aligned} \varepsilon \parallel y = y \parallel \varepsilon &= \{y\} \\ xa \parallel yb &= (x \parallel yb)\, a \cup (xa \parallel y)\, b. \end{aligned}$$

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.

Determinism

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.

Problem 2: Determinism and Union

Consider the binary rational relations

$$A = \binom{aa}{b}^{\star} \qquad B = \binom{a}{bb}^{\star}$$

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

Now consider

$$A \cup B = \{ \binom{a^i}{b^j} \mid i = 2j \vee j = 2i \}$$

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).

Word Orders: Length Order

Consider the binary relation $<_{\mathrm{len}}$ on $\Sigma^\star$ defined by

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

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 $<_{\mathrm{s}}$ on $\Sigma^\star$ defined by

$$x <_{\mathrm{s}} y \iff \exists\, a < b \in \Sigma, u, v, w \in \Sigma^\star\, (x = uav \wedge y = ubw)$$

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

Lexicographic Order

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

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

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.

Length-Lex Order

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

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

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.

Concatenation is 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 $\Sigma = \{a, b\}$

$$\gamma = (a{:}\varepsilon{:}a + b{:}\varepsilon{:}b)^\star \cdot (\varepsilon{:}a{:}a + \varepsilon{:}b{:}b)^\star$$

□

Addition is Rational

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

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

where $\mathrm{bin}(x)$ 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).

Relational Composition

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

Proof

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

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

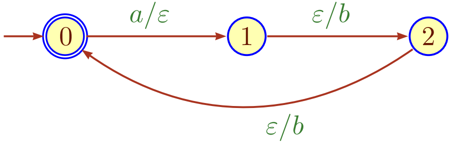
whenever there are transitions $p \xrightarrow{a/b} p'$ and $q \xrightarrow{b/c} q'$ in $\mathcal{A}$ and $\mathcal{B}$, respectively, for some $a \in \Sigma_\varepsilon$, $b \in \Gamma_\varepsilon$ and $c \in \Delta_\varepsilon$.

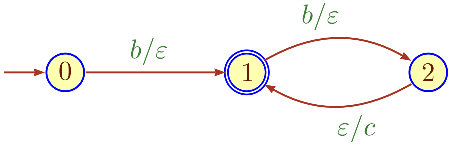
Initial and final states in $\mathcal{C}$ are $I_1 \times I_2$ and $F_1 \times F_2$. It is a labor of love to check that $\mathcal{C}$ accepts x/z if, and only if, $x \,\rho\, y$ and $y \,\sigma\, z$ for some $y \in \Gamma^\star$. $\square$

Example

Let $\rho = \binom{a}{bb}^\star$ and $\sigma = \binom{b}{\varepsilon}\binom{b}{c}^\star$; thus $\rho \circ \sigma = \binom{a}{c}\binom{a}{cc}^\star$.

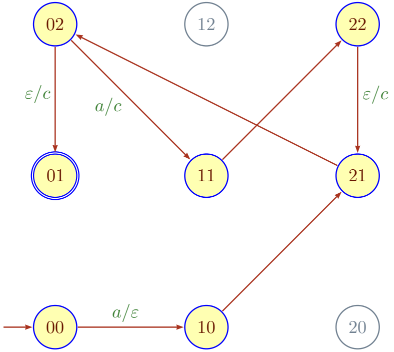
Here are the two machines, without the ε/ε self-loops.





Example

And here is the accessible part of the product. Unlabeled edges are supposed to be ε/ε.



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

Projections

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

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

Lemma

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

Proof.

Erase the first track in the k-track alphabet:

$$p \xrightarrow{a_1:a_2:\ldots:a_k} q \quad \rightsquigarrow \quad p \xrightarrow{a_2:\ldots:a_k} q$$

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

It's a Clone

Note that the use of the term *projection* is slightly different here from the standard use: $\boldsymbol{x} \mapsto x_i$.

Clearly, rational relations contain ordinary projections in this sense.

So, we are really dealing with a clone, except that this time we have a clone of relations rather than a clone of functions (recall the section on computability).

And while we are talking about bad terminology and notation ...

Transitive Closure

One might wonder what happens when we move to the transitive reflexive closure tcl(ρ). Recall that

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

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 $\rho^\star$, 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, $\mathsf{tcl}(\rho) = \rho$.

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

Transitive Closure is Semidecidable

Theorem

The transitive closure tcl(ρ) *of a rational relation is semidecidable.*

Proof.

By definition x tcl(ρ) 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. $\square$

Semidecidability

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.

□

Arithmetical Transducers

If we write natural numbers as binary string, arithmetical operations turn into transductions.

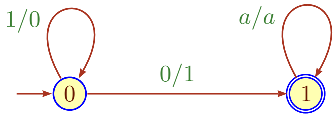
Note that one really needs to fix a numeration system to make this more precise.

- LSD left or right.
- Any string acceptable.
- No empty string.
- No trailing zeros.

A reasonable convention would be the numeration system $\mathcal{N}$: LSD first (reverse binary), no empty string, no trailing zero.

Successor

The key component for the successor trunsduction is the following machine:

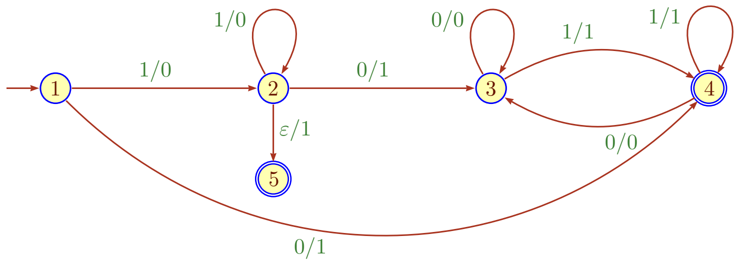


Alas, this allows for trailing zeros. Worse, it implements a cyclic counter: $1^k \mapsto 0^k$.

To fix things, it can help to write down a list of specifications for a transducer that is compliant with the numeration system $\mathcal{N}$.

$$\begin{array}{lcl} 0\,x & \rightsquigarrow & 1\,x \\ 1\,1^k & \rightsquigarrow & 0\,0^k\,1 \\ 1\,1^k\,0\,x & \rightsquigarrow & 0\,0^k\,1\,x \end{array}$$

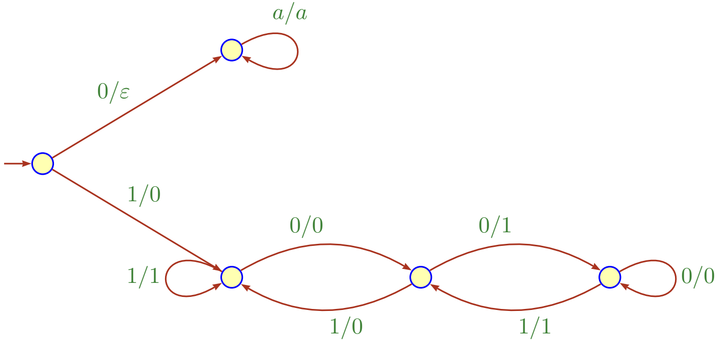
Compliant Successor



This version is obtained by a bit of surgery on the basic machine, and is compliant with our numeration system $\mathcal{N}$.

For correctness, one can show by induction that it implements the specifications from the last slide.

A Holy Terror



Collatz

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.

Multiplication

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.

Here is a general recommendation: organize the construction of the transducer into 3 phases.

1. Consider the simplified situation where the input is infinitely long, so there is no need to worry about what to do when the input ends.
2. Handle the case when the input actually ends.
3. Worry about endmarkers and padding symbols.

Getting everything correct with endmarkers right from the start is hard, it's much easier to work in stages.

Example: Multiply by 5, in Binary

As always, the key is to pick the right state set. We can add the strings x and $00x$ to get multiplication by 5.

$$\begin{array}{c|cccccccccccc} x & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & 0 & 0 & 0 & 0 & \ldots \\ 00x & 0 & 0 & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & 0 & 0 & \ldots \end{array}$$

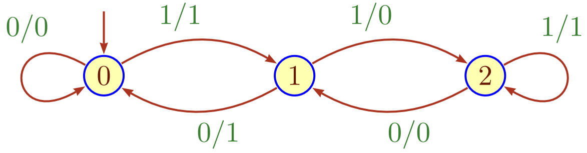
So we really have to deal with addition, using the standard add digits and keep track carries approach. But note that there is only one input, and we have to remember the current input bit for two more steps.

Exercise

Explain how to construct the multiply-by-5 transducer.

Multiply-by-3

Here is the "multiply-by-3" transducer, $B = 2$, $m = 3$ for reverse binary.



Note that the input must be padded $x \mapsto x00$, otherwise we fail to take the carry into account. Unfortunately, this can lead to trailing 0s, which is a bit clumsy (see below). Lastly, this machine interprets ε as 0, not necessarily a good idea.

Input/Output Functions

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

The multi-word accepted by a run from p to q is then modified from $u{:}v$ to

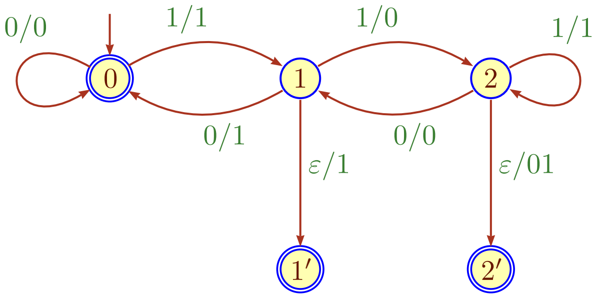
$$u : \mathsf{inp}(p)\, v\, \mathsf{outp}(q)$$

For full-fledged transducers this makes no real difference, we could simply add corresponding transitions with ε-labels. For limited classes of transducers, such as alphabetic ones, this convention is very convenient.

Exercise

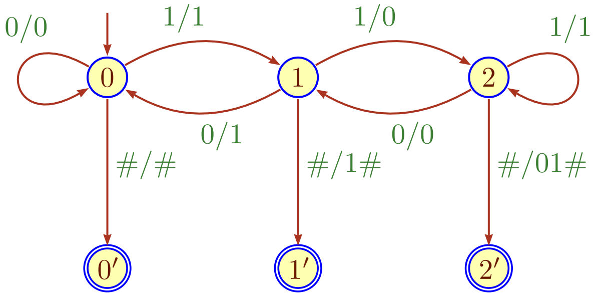
Figure out how to add an output map to the "multiply-by-3" transducer to fix it (no input padding needed).

Example: Multiply-by-3



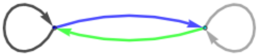
Here we only need output functions, depending on the carry that is associated with each state.

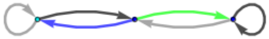
Example: Multiply-by-3

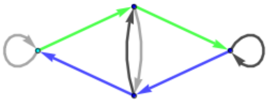


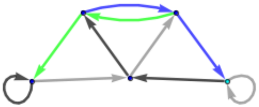
Lastly, a machine that assumes that every track has a special endmarker $\#$. Note that this machine is clearly deterministic, the endmarkers tell us when the end of the input has been reached.

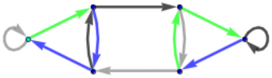
$B = 2$, $m = 2, \ldots, 7$

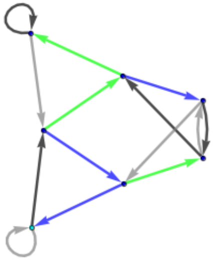












Length-Preserving Iteration

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.*

So the problem is decidable, but the algorithm may blow up.

Fixed-Points

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 Turing machine without input, operating on bounded tape. Set things up so that all orbits end in a fixed point iff the Turing 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.

$\square$

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

Scaling Back

Rational relations in general are just a little too powerful for our purposes, they don't have nice closure properties (the way regular languages do). 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 $\rho \subseteq (\Sigma \times \Gamma)^\star$, so our multi-words are actually 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 $\Sigma \times \Gamma$.

Aside: Mealy Machines

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).

Relaxing the Length Condition

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 written $x{:}y^\#$.

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

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

Comment 1

Note that we are not using all of $\Delta_{\#}^{\star}$ but only the regular subset coming from convolutions. In other words, $\#$ can only appear as a suffix, and in exactly one track. For example,

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

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

are not allowed.

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

$$\Delta_{\#} = \Sigma_{1,\#} \times \Sigma_{2,\#} \times \ldots \times \Sigma_{k,\#}$$

Exercise

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

Comment 2

It may be preferable to terminate all tracks with a # symbol (so that the corresponding machines have a unique final state).

$x{:}y^{\#e} =$

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

There is no essential difference between the two versions, there are pro and cons for both approaches.

Synchronous Relations

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

Definition

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

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

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.

(Counter)Examples

- Equality and inequality are synchronous.
- 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 (scattered) subword of y" are not synchronous.

The Difference?

Intuitively, the difference between arbitrary transductions and synchronous ones is that, for the latter, one can build a 2-track machine whose heads can move independently, but are never further than some fixed distance d apart.

Bounded head distance is already enough: essentially, we could then force phantom heads to move in lockstep by remembering the last d symbols and the actual head positions.

So the critical difference is when the two heads move arbitrarily far away from each other.

A Justification

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.

Boolean Operations

Claim

Given two k-ary synchronous relations ρ and σ on $\Sigma^\star$, the following relations are also synchronous:

$$\rho \sqcup \sigma \qquad \rho \sqcap \sigma \qquad \rho - \sigma$$

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).

Warning: Concatenation

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

$$\rho = \binom{a}{\varepsilon}^\star \binom{\varepsilon}{a}^\star$$
$$\sigma = \binom{b}{b}^\star$$

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

Exercise

Prove all examples and counterexamples.

Synchronous Composition

On the upside, synchronous relations are closed under composition.

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

Theorem

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

Exercise

Prove the theorem.

Synchronous Projections

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.

A Simple Relational Structure

To simplify matters, suppose we are looking at a structure over the alphabet $\mathbf{2}$ with just one binary relation:

$$\mathfrak{C} = \langle \mathbf{2}^\star, \twoheadrightarrow \rangle$$

In fact, $\twoheadrightarrow$ could be a function, interpreted as a binary relation. Think about elementary cellular automata, say, with periodic boundary conditions. Strictly speaking, the carrier set should be $\mathbf{2}^+$.

Just to be clear: this scenario includes Turing machines, we could easily code up the one-step relation over a binary alphabet. But then the relation is not length preserving and things get much more complicated.

Note, though, that we cannot ask questions about orbits, first-order logic is simply too weak for that.

Examples

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

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

$$\forall\, x\, \exists\, y\, (y \rightarrowtriangle x)$$

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

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

Meaning: single-valued, injective, surjective, 3-cycle, indegree 2.

The Model Checking Algorithm

So suppose we have the finite state machines describing $\mathfrak{C} = \langle \mathbf{2}^\star, \twoheadrightarrow \rangle$ 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:

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

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

[†]This is actually a bad idea for efficiency reasons, but it simplifies the discussion of the basic algorithm.

Atomic

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

- $x_i = x_j$
- $x_i \rightarrowtriangle x_j$

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

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

From Atomic to Quantifier-Free

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 $\rightarrowtriangle$ or $=$.

Note that there is a bureaucratic problem: the atomic machines are 2-track, but the machine for the matrix is usually k-track for some $k > 2$.

Embeddings

More precisely, use superscripts to indicate the number of tracks of a machine as in $\mathcal{A}^{(2)}_{\rightarrow}$ and $\mathcal{A}^{(2)}_{=}$.

Let $m \leq n$. We need an embedding operation

$$\mathsf{emb}^{(n)}_{\boldsymbol{t}} : m\text{-track} \longrightarrow n\text{-track}$$

where $\boldsymbol{t} = t_1, \ldots, t_m$, $t_i \in [n]$, all distinct.

So $\mathsf{emb}^{(n)}_{\boldsymbol{t}}(\mathcal{A}^{(m)}) = \mathcal{B}^{(n)}$ means that track i of $\mathcal{A}^{(m)}$ is identified with track t_i in $\mathcal{B}^{(n)}$. The other tracks are free (all possible transitions). This does not affect the state set, but it can cause potentially very large alphabets and, correspondingly, large numbers of transitions in the embedded automaton[†].

[†]One of the reasons why state complexity alone is not really a good measure of the size of an automaton, one needs to add the number of transitions.

Padding

Another issue to be careful with is padding. Suppose we have a 2-track machine $\mathcal{A}^{(2)}$ and we want to construct the product machine

$$\mathcal{B} = \mathsf{emb}^{(3)}_{1,2}\big(\mathcal{A}^{(2)}\big) \times \mathsf{emb}^{(3)}_{2,3}\big(\mathcal{A}^{(2)}\big)$$

Since the word lengths in the 3 tracks may be different, we have to temporarily allow for multiple trailing $\#\#$ symbols.

This is particularly simple in the endmarker variant: just add a self-loop to the single final state reached via input $\#\#$:

$$p \xrightarrow{\#\#} q \xrightarrow{\#\#} q$$

Boolean Connectives

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

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 $\mathcal{A}_\varphi$ that recognizes the relation defined by the matrix:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{\, u_1{:}u_2{:}\ldots{:}{u_k}^{\#} \mid \mathcal{C} \models \varphi(u_1, u_2, \ldots, u_k) \,\}$$

Projections

There is a natural dual to embeddings: projections.

Let $m \leq n$. We have a projection operation

$$\mathsf{prj}_{\boldsymbol{t}}^{(n)} : n\text{-track} \longrightarrow m\text{-track}$$

where $\boldsymbol{t} = t_1, \ldots, t_{n'}$, $n' \leq n$, $t_i \in [n]$, all distinct, $m = n - n'$.

So $\mathsf{prj}_{\boldsymbol{t}}^{(n)}(\mathcal{A}^{(n)}) = \mathcal{B}^{(m)}$ means that, for all transitions in $\mathcal{A}^{(n)}$, the tracks t_i of the transition labels have been erased, producing $\mathcal{B}^m$. The state set is unaffected.

It is fine to have $n = n'$, in which case it is understood that we are left with an unlabeled digraph (with special initial and final nodes).

Quantifiers

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 $\mathcal{A}_{\psi}^{(n)}$ that has a track t for variable x.

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

In other words, $\mathrm{prj}_t^{(n)}(\mathcal{A}^{(n)})$ corresponds exactly to existential quantification over variable x.

Alas, for universal quantifiers we have to 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.

Finale Furioso

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

$$\mathcal{L}(\mathcal{B}_{\varphi,\ell}) = \{\, u_1{:}u_2{:}\ldots{:}u_\ell{}^{\#} \mid \mathcal{C} \models \varphi_\ell(u_1, u_2, \ldots, u_\ell) \,\}$$

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.

Closure to the Rescue

Why does this all work, fundamentally?

$\vee$	closure union
$\wedge$	closure intersection
$\neg$	closure complement
$\exists$	closure homomorphism
emb	closure inverse homomorphism

And not just closure, but effective closure: we have algorithms to construct all the corresponding machines.

Efficiency

- $\vee$ 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.
- $\forall$ well . . .

So this is a bit disappointing: we may run out of computational steam even when the formula is not terribly large. Universal quantifiers, in particular, can be a major problem.

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

Example: 3-Cycles

Let's figure out the details on how to determine the existence of a 3-cycle in $\mathcal{C}$. The obvious formula to use is this:

$$\Phi \equiv \exists\, x, y, z\, (x \rightarrowtriangle y \land y \rightarrowtriangle z \land z \rightarrowtriangle x \land x \neq y \land x \neq z \land y \neq z)$$

The first part ensures that there is a cycle, and the second part prevents the cycle from being shorter than 3.

Perfectly correct, but note the following. Suppose the basic machine $\mathcal{A}_{\rightarrowtriangle}$ that checks $\rightarrowtriangle$ has m states. Then the first part of the formula produces a machine of possibly m^3 states. The non-equal part blows things up further to $8\, m^3$ states.

Optimizing

We could replace Φ by any equivalent formula, which would be usefully if we could find a smaller formula. It seems hard to get around the m^3 part, 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 better to realize that since $\rightarrowtriangle$ is functional, the last formula is equivalent to

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

Exercise

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

The 3-Cycle Transducer

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

$$\{\, u{:}v{:}w^{\#} \mid u \twoheadrightarrow v \twoheadrightarrow w \twoheadrightarrow u \wedge u \neq v \,\}$$

Let $\mathcal{A}_{i,j} = \mathsf{emb}^{(3)}_{i,j}\big(\mathcal{A}^{(2)}_{\twoheadrightarrow}\big)$. Also, let $\mathcal{D}^{(2)}_{\neq}$ be the machine that checks for inequality and $\mathcal{D} = \mathsf{emb}^{(3)}_{1,2}\big(\mathcal{D}^{(2)}_{\neq}\big)$.

We can now concoct a 3-track product machine for the conjunctions:

$$\mathcal{B} = \mathcal{A}_{,1,2} \times \mathcal{A}_{2,3} \times \mathcal{A}_{3,1} \times \mathcal{D}$$

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

Projecting

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

Once $\mathcal{B}_3$ 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 $\rightarrowtriangle$.

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

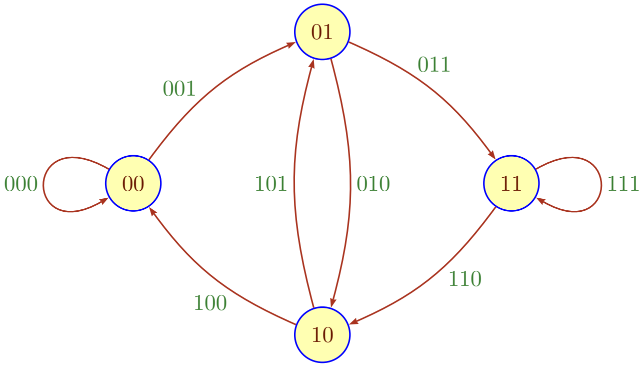
Example: ECA

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

It turns out to be a little easier if we first consider configurations over $\mathbf{2}^{\mathbb{Z}}$. Finite is often harder than infinite.

First, an automaton that corresponds to sliding a window of length 2 across the configuration. The states will naturally be $\mathbf{2}^2$, and the edges corresponds to just having seen 3 bits in a row.

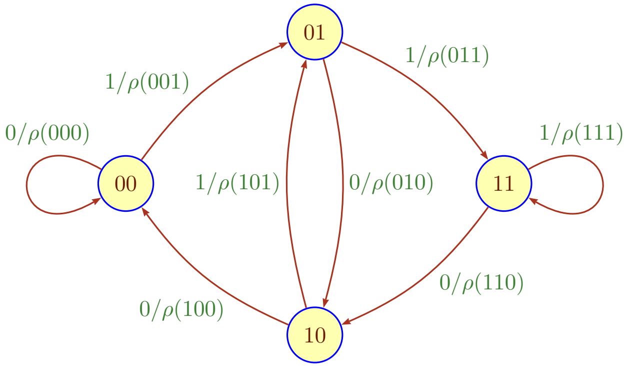
The de Bruijn Automaton



$$\begin{tikzcd} & 01 \arrow[dr, bend left, "011"] \arrow[d, bend left, "010"] & \\ 00 \arrow[loop left, "000"] \arrow[ur, bend left, "001"] & 10 \arrow[l, bend left, "100"] \arrow[u, bend left, "101"] & 11 \arrow[loop right, "111"] \arrow[l, bend left, "110"] \end{tikzcd}$$

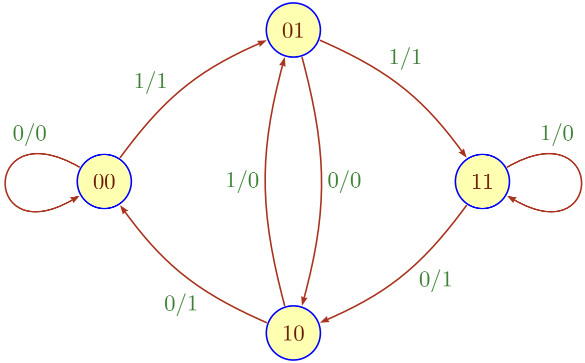
Each configuration in $\mathbf{2}^{\mathbb{Z}}$ corresponds to exactly one biinfinite path in this automaton. And every biinfinite path corresponds to a configuration (at least if we are a bit relaxed about where the origin is).

The Basic Transducer



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.

Example: Rule 90



A rather civilized transducer: a partial DFA over the alphabet $\mathbf{2}^2$. It remains deterministic even if we project away either one of the tracks.

Recall: Boundary Conditions

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 $\boldsymbol{x} = x_1 x_2 \ldots x_n$ with FBC we work on $0 x_1 x_2 \ldots x_n 0$ leading to n many 3-blocks

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

that we apply the local map f to.

The Finite Case

The last transducer works on $\mathbf{2}^{\mathbb{Z}}$, but how about plain $\mathbf{2}^n$? 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.

Elgot/Mezei to the Rescue

If we drop the synchronicity condition, there is no problem: it easy to see that $\rightarrowtriangle$ is rational. And $\rightarrowtriangle$ 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 $\rightarrowtriangle$ is rational.

A Synchronous Transducer

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

Automaton $\mathcal{A}_{\rightarrowtriangle}$ 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 $\mathbf{2}^{\mathbb{Z}}$.

A Computation

input	state	condition
$-$	$\perp$	$-$
$u_1{:}v_1$	$0\,u_1u_2$	$v_1 = \rho(0u_1u_2)$
$u_2{:}v_2$	$u_1u_2u_3$	$v_2 = \rho(u_1u_2u_3)$
$u_3{:}v_3$	$u_2u_3u_4$	$v_3 = \rho(u_3u_3u_4)$
	$\vdots$	
$u_{n-1}{:}v_{n-1}$	$u_{n-2}u_{n-1}u_n$	$v_n = \rho(u_{n-1}u_n0)$
$u_n{:}v_n$	$\top$	$-$

A successful computation on input $u_1u_2\ldots u_n{:}v_1v_2\ldots v_n$.

Tricks

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 (e.g., the antichain method that avoids direct determinization).

More Tricks

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 large formula that defines some property (which formula is then is translated into a potentially very large automaton), we just build the automaton directly from scratch 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.

Multiply by 5

Here is a simple example: suppose we want to construct a transducer that multiplies by 5 (numbers in reverse binary).

We already have 2-track machines $\mathcal{M}_2^{(2)}$ and $\mathcal{M}_3^{(2)}$ that multiply by 2 and 3, respectively. We also have a 3-track adding machine $\mathcal{A}_{\mathsf{add}}^{(3)}$.

We can assemble the new multiplier from these pieces:

$$\mathcal{B} = \mathsf{emb}_{1,2}^{(4)}\big(\mathcal{M}_2\big) \times \mathsf{emb}_{1,2}^{(4)}\big(\mathcal{M}_2\big) \times \mathsf{emb}_{2,3,4,}^{(4)}\big(\mathcal{A}_{\mathsf{add}}\big)$$

$$\mathcal{M}_5 = \mathsf{prj}_{2,3}^{(2)}\big(\mathcal{B}\big)$$

But, we can also build $\mathcal{M}_5$ directly, by finding a suitable abstract description of the the transitions.