CDM
Automatic Structures and FOL

Klaus Sutner

Carnegie Mellon Universality

92-elem-ca 2017/12/15 23:20

Disclaimer

Recall that we want to use automatic structures as an example for model checking.

Alas, real world applications are quite messy, so we will avoid them like the plague and instead apply our machinery to simple systems from discrete dynamics, so-called one-dimensional cellular automata. And, we will only deal with first-order logic (though many applications in CS require e.g. temporal logic).

These have the advantage that they are very easy to explain and lead to manageable problems: one can actually understand the finite state machines that pop up in the decision algorithm.

And they are quite interesting as an independent subject of study, sitting at the intersection of symbolic dynamics, computability and automata theory.

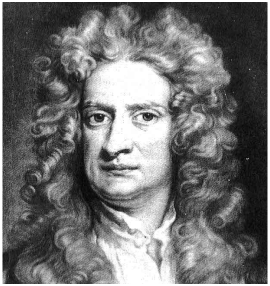
The Real World

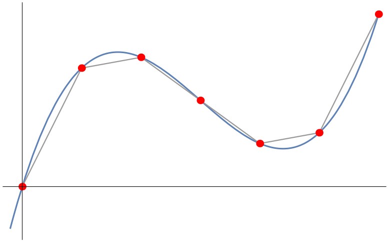
J. Harrison.
Handbook of Practical Logic and Automated Reasoning.
Cambridge University Press, 2009.

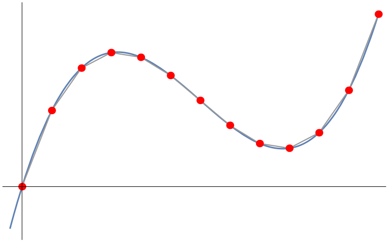
E. Clarke, O. Grumberg, and D. Peled.
Model Checking.
MIT Press, 2000.

D.B.A. Epstein, J.W. Cannon, D.F. Holt, S.V.F. Levy, M.S. Patterson, and W.P. Thurston.
Word Processing in Groups.
Jones and Bartlett, Burlington, 1992.

The Classical World







That’s all, folks.

Dynamical Systems

- Variables (bunch of real numbers)
- Equations of motion (typically (partial) differential equations)

Get phase space: describes evolution over time.

Good systems are "integrable," smooth, very tame and well-behaved. This is an example of computational compressibility: one can figure out what happens later without having to run the system.

Hugely successful enterprise, (almost) all of 20th century physics, including relativity theory and quantum physics, depends on this.

Pathologies

Smooth functions are obviously important to describe reality, but in a way they are really bizarre exceptions: they form a set of measure zero in the right setting.

So if you pick a continuous function at random, with probability 1, it will be nowhere differentiable. Calculus will be entirely useless to analyze this function.

Some would say that these functions, like their discoverers, are just pathological; other than some deranged pure mathematicians and logicians, no one wastes any time thinking about this nonsense.

H. Poincaré

Formerly, when one invented a new function, it was to further some practical purpose; today one invents them in order to make incorrect the reasoning of our fathers, and nothing more will ever be accomplished by these inventions.

Ouch.

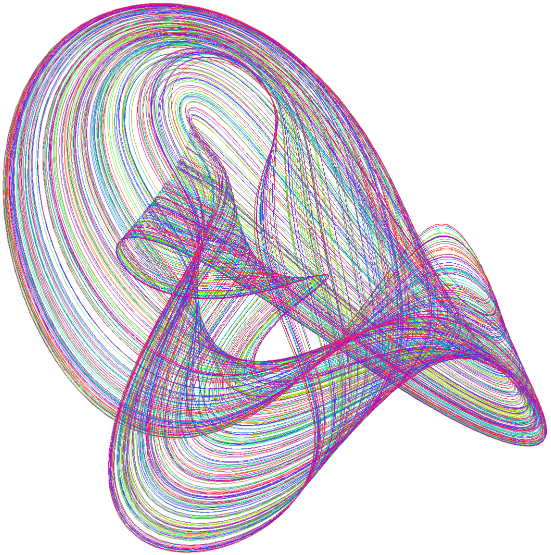
Poincaré was not very fond of the emerging fields of set theory and logic. And, of course, he was brilliant.

Lorenz

Here is a system that was discovered by E. Lorenz in 1963, in an attempt to model atmospheric convection. His stripped-down equations are embarrassingly simple (σ, ρ and β are parameters; these are plain ODEs):

$$\begin{aligned} x' &= \sigma(y-x) \\ y' &= x(\rho-z)-y \\ z' &= xy-\beta z \end{aligned}$$

Someone with experience in computability theory might be a bit cautious: even extremely simple systems can behave in a very complicated fashion under iteration (computationally universal).



Don't Pretend

Any sensible theory of physics must respect these limitations, and should not invoke calculative routines that in fact cannot be carried out.

R. Landauer (1927-1999)

M. Tegmark.
Our Mathematical Universe.
Alfred A. Knopf, 2014.

Tegmark very much enjoys making controversial statements and getting people all riled up; unfortunately, this particular book overshoots by a few lightyears.

The central claim: all mathematical structures correspond to physical reality. Of course, Tegmark is a physicist, so his idea of a mathematical structure is heavily biased – but still.

Automatic Structures

Recall that a relational structure is a FO structure of the form

$$\mathcal{C} = \langle\, A; R_1, R_2, \ldots, R_k \,\rangle$$

where the R_i are all relations (there are no functions).

Definition

A automatic structure is a relational structure where the carrier set $A \subseteq \Sigma^\star$ is regular and all the relations R_i are automatic (aka synchronous).

Automatic is Easy

Theorem

Model checking for automatic structures and first-order logic is decidable.

An automatic structure $\mathcal{C}$ is given by

- an acceptor $\mathcal{A}$ that recognizes A, and
- synchronous transducers $\mathcal{A}_i$ for R_i (so really just acceptors over the padded alphabet).

So, all we have is a list of finite state machines, a perfectly good data structure.

Discrete Phase Space

How do we squeeze an automatic structure out of dynamics?

Here are a few radical assumptions that bring the study of dynamical systems into the computationally harmless realm of automata theory:

- Time is discrete.
- Space is discrete and one-dimensional.
- Properties are discrete.

This conforms nicely to the idea that a finite amount of information should be enough to describe a bounded region of spacetime.

The Standard Model

We think of space as a grid of discrete locations, which locations are often called cells. If we assume static topology, we get something like $C = \mathbb{Z}^d$ or $C = \mathbb{N}^d$ or $C = [n]^d$.

Time is discrete, something like $\mathbb{Z}$ or $\mathbb{N}$ or $[n]$.

At each moment in time each cell is associated with one of a finite set Σ of possible values that describes the local conditions at this point in spacetime. A configuration at time t is a map $C \to \Sigma$.

Lastly, there is an evolution operator $\to$ that determines the configuration at time $t+1$, given the configuration at time t.

Back to First-Order

So we can model a discrete phase space as a FO structure

$$\mathfrak{C} = \langle C, \to \rangle$$

where we think of the evolution operator as a binary relation: $x \to y$ means that configuration x evolves to configuration y in one step.

We will focus on the case where $\to$ is deterministic (functional), and, more importantly, the space is one-dimensional.

The Key Idea:
The space of configurations C is a regular set of words and $\to$ is a synchronous relation on C.

Some Typical Dynamical Systems Questions

The following questions are all quite natural:

- Is the system reversible?
- Does every state have a predecessor?
- Does every state have exactly k predecessors?
- Does every state have exactly k successors?
- 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 given initial state p?

A Good Framework

We are only dealing with first-order logic, so some of these questions are off-limits. In fact, they turn out to be undecidable, even if $\to$ is synchronous. But others can be handled nicely (at least ignoring computational complexity for the time being).

For example

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

means that every state has a predecessor state.

And

$$\forall\, x\, \exists\, y\, (x \to y \land y \to y)$$

means that every configuration evolves to a fixed point in (at most) one step.

More Examples

Assuming $\to$ is functional, no point is a fixed point:

$$\forall\, x\, \exists\, y\, (x \to y \wedge y \neq x)$$

The system is reversible ($\to$ is injective):

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

Every point has exactly two distinct predecessors:

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

Elementary Cellular Automata

Here is the kind of discrete dynamical system we are going to analyze: one-dimensional, binary, continuous global behavior.

For the grid of cells we have the basic options

$C = \mathbb{Z}$	biinfinite
$C = \mathbb{N}$	one-way infinite
$C = [n]$	finite

The first two are studied the classical setting, but simulations always work with finite configurations.

The Laws

Definition

A local map or local rule is a function

$$\rho : \mathbf{2}^3 \longrightarrow \mathbf{2}$$

Thus, a local map is just a ternary Boolean function. Note that a local map can be specified by 8 bits, one for each 3-bit input.

So, there are exactly $2^{2^3} = 256$ local maps.

This is a good number: one can easily check them all out, but there might just be enough of them for something interesting to happen.

Global Maps

Definition

Let $\mathbf{2}^\infty = \mathbb{Z} \to \mathbf{2}$ denote the collection of all (biinfinite) configurations.

We can extend any local map ρ to a global map

$$G_\rho : \mathbf{2}^\infty \longrightarrow \mathbf{2}^\infty$$

by setting

$$G_\rho(X)(i) = \rho(X(i-1), X(i), X(i+1))$$

Note that $\mathbf{2}^\infty$ is a wildly uncountable space, though it has lots of interesting properties (compact, zero-dimensional, totally disconnected Hausdorff space).

Boundaries

Alas, this does not work for $C = \mathbb{N}$: there is a cell that does not have a left neighbor, so our definition makes no sense there. This is easy to fix:

- Fixed boundary conditions: assume a phantom cell clamped in state 0.

A similar problem occurs for $C = [n]$, now 2 cells have missing neighbors. This time there are two solutions:

- Fixed boundary conditions: assume two phantom cells clamped in state 0.
- Cyclic boundary conditions: assume the grid wraps around (circle of cells).

As it turns out, the two types of boundary conditions usually lead to very similar behavior.

Experimental Evidence

We would like to understand the global map G_ρ induced by each of the 256 local rules ρ.

An excellent way to get a first impression is to compute a (small initial segment) of the orbit of some configuration X under G_ρ.

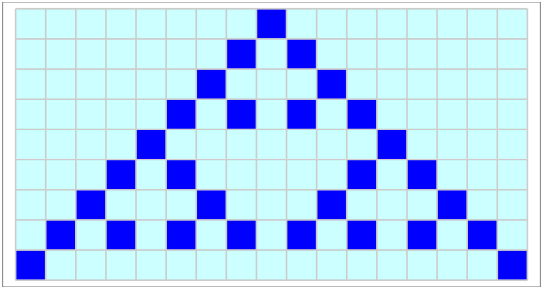
Of course, we can only do this in the finite case. But that is quite alright, the dynamics on infinite configurations are well reflected in the finite scenario.

Orbits and Pictures

Applying G_ρ t times to x we get a sequence of bit-vectors of length n.

We can get a simple picture by stacking the bit-vectors to a Boolean matrix of dimension $(t+1) \times n$. Nothing is easier to plot than a Boolean matrix.

For example, local rule $\rho(x, y, z) = x \oplus z$ on a 1-point seed configuration produces



Live demo.

A Relational Structure

So we are looking at a structure

$$\mathfrak{C} = \langle C, \to \rangle$$

where $\to$ is just G_ρ, but interpreted as a binary relation.

We need to fix boundary conditions once and for all. Say, we only consider fixed boundary conditions.

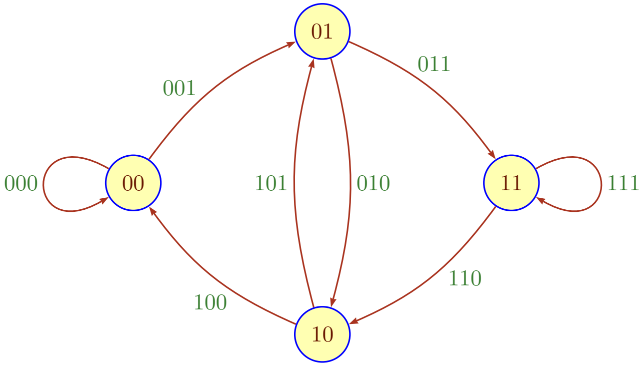
The first problem is to deal with atomic formulae, in particular

$$x \to y$$

We need a transducer $\mathcal{A}_\to$ that checks that x evolves to y in one step under $\to$.

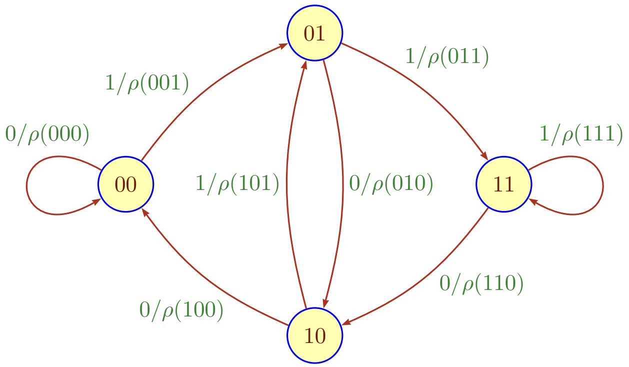
given words u and v. Note that $\to$ is trivially length-preserving, so we don't have to bother with endmarkers.

The de Bruijn Automaton



The main idea is easy: move a window of width 2 across x. Each time the window advances we have full information about a 3-bit block in x, here written as edge labels.

The Basic Transducer



If we replace the edge labels xyz by $xyz/\rho(xyz)$ we get a transducer that simulates the cellular automaton. All states are initial and final, we are interested in biinfinite runs.

Note that we ultimately want a synchronous transducer.

Wait a Moment

OK, so we are cheating a little bit: this idea works perfectly well if we are dealing with the classical scenario of

$$\mathbf{2}^{\infty} = \mathbb{Z} \to \mathbf{2}$$

as carrier set.

Alas, we don't yet have a framework for automata on infinite words, we really need to deal with the finite case.

Because of boundary conditions, things get slightly more complicate. For the time being, assume fixed boundary conditions.

The Finite Case

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, 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 $\rightarrow$ 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.

A Synchronous Transducer

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 $Q = \{\bot, \top\} \cup \mathbf{2}^3$.

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

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

Exercises

Exercise

Prove that the automaton from the last slide really works.

Exercise

Build some of the transducers $\mathcal{A}_{\rightarrow}$ and minimize them.

Exercise

Show how to define the analogous transducer when periodic boundary conditions are used. Prove that your construction works.

Atomic

In terms of a FOL formula, we now can deal with the atomic cases

- $x_1 = x_2$
- $x_1 \to x_2$.

using two synchronous transducers $\mathcal{A}_=$ and $\mathcal{A}_\to$.

Since our structures are purely relational, we do not have to worry about general terms, there is no such thing as $R(f(x), y)$.

From Atomic to Quantifier-Free

Suppose we have a quantifier-free 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 atomic pieces read from two appropriate tracks and check $\to$ or $=$.

More precisely, we use variants $\mathcal{A}_{\to,i,j}$ 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.

Everything else is just Boolean combinations and we wind up with a composite automaton $\mathcal{A}_\varphi$

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

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

$$\exists\, x_1\, \exists\, x_2\, \forall\, x_3\, \exists\, x_4\, \forall\, x_5\, \ldots \varphi(\boldsymbol{x}, \boldsymbol{z})$$

where φ is quantifier-free, $\boldsymbol{x}$ denotes the quantified variables and $\boldsymbol{z}$ stands for the free variables. Indeed, there is a simple algorithm to convert an arbitrary formula to PNF.

So we start with an automaton, constructed from the basic ones by Boolean combinations, with $|\boldsymbol{x}| + |\boldsymbol{z}|$ tracks. Then we eliminate all the quantified tracks, dealing with universal quantifiers via $\forall \equiv \neg\exists\,\neg$ and with existential ones by projection.

This is admissible, since projection does not affect automaticity.

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

Finale Furioso

If φ has free variables $z_1, \ldots, z_\ell$ we wind up with an ℓ-track automaton $\mathcal{A}_\varphi$ that accepts precisely those ℓ-track words that satisfy the formula:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{\, u_1{:}u_2{:}\ldots{:}u_\ell \in \Gamma_k^\star \mid \mathfrak{C} \models \varphi(u_1, u_2, \ldots, u_\ell) \,\}$$

In particular, 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.

So in the end, it all comes down to path existence and the last test is linear. Of course, there is a bit of work to get there ...

Efficiency

- $\lor$ and $\exists$ are linear if we allow nondeterminism.
- $\land$ 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.

Spectra

It should not be a surprise that the behavior of the restricted map

$$G_\rho : \mathbf{2}^n \to \mathbf{2}^n$$

in general depends on n.

This suggests a version of data model checking: we fix a specification φ and determine all n for which φ holds. This is called the spectrum of φ.

$$\mathsf{spec}(\varphi) = \{\, n \in \mathbb{N} \mid \mathfrak{C}_n \models \varphi \,\}$$

Here $\mathfrak{C}_n = \langle \mathbf{2}^n, \to \rangle$ is the automatic structure built from all bit-vectors of length n only.

Written in unary, the spectrum is a regular language and thus ultimately periodic (or semi-linear, if you prefer algebraic language).

3-Cycles

As an example how this might work, consider the question of whether there is a 3-cycle under G_ρ on $\mathbf{2}^n$. The brute-force version of the sentence φ looks like

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

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 the last formula is equivalent to

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

Exercise

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

The 3-Cycle Transducer

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

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

$$\mathcal{C}_3 = \mathcal{A}_{\to,1,2} \times \mathcal{A}_{\to,2,3} \times \mathcal{A}_{\to,3,1} \times \mathcal{U}_{1,2}$$

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

Machine $\mathcal{U}_{i,j}$ tests if the word in track i is different from the word in track j and consists essentially of two copies of $\mathcal{A}_{\to,1,2} \times \mathcal{A}_{\to,2,3} \times \mathcal{A}_{\to,3,1}$.

Projecting

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 free variables everything is projected away).

This digraph has a path of length n from an initial state to a final state if, and only if, there is a 3-cycle under G_ρ on $\mathbf{2}^n$.

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.

Disclaimer I

Note that cellular automata exist in arbitrary dimensions $d \geq 1$.

Unfortunately, this machinery applies exclusively to dimension 1: there is no d-dimensional analogue of a finite state machine (more precisely, these analogues don't have the right properties).

It is known that even injectivity and surjectivity of G_ρ is undecidable for 2-dimensional cellular automata, never mind the full first-order theory.

Disclaimer II

The machinery we just discussed for the finite case also applies, mutatis mutandis, to the two infinite scenarios: the carrier sets are uncountable, but the first-order theory is still decidable.

But: for this to work we have to generalize finite state machines to operate on infinite words, and that takes a bit of effort.

In particular determinization of these generalized machines is a major nightmare.

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.

More Tricks

We can easily augment our decision machinery by adding 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.