CDM

Presburger Arithmetic

K. Sutner

Carnegie Mellon University

Fall 2024

Logic is Hard

> Everybody who has worked in formal logic will confirm that it is one of the technically most refractory parts of mathematics. The reason for this is that it deals with rigid, all-or-none concepts, and has very little contact with the continuous concept of the real or of the complex number, that is, with mathematical analysis. Yet analysis is the technically most successful and best-elaborated part of mathematics. Thus formal logic is, by the nature of its approach, cut off from the best cultivated portions of mathematics, and forced onto the most difficult part of the mathematical terrain, into combinatorics.
>
> John von Neumann, 1948

1 **Numeration Systems**

2 **Presburger Arithmetic**

3 **Automatic Structures**

4 **Deciding Presburger Arithmetic**

Numeration Systems

A numeration system is a method to denote all natural numbers by words over a digit alphabet $\varDelta$.

The digits all directly correspond to particular numbers, typically 0, 1, 2, -1 and so forth. In a positional notation system the numerical value of a digit string $d = d_0 d_1 \ldots d_{k-1}$ is determined by weights $b_i \in \mathbb{N}_+$, $i \geq 0$, as follows. The value map val : $\varDelta^\star \to \mathbb{N}$

$$\mathsf{val}(d) = \sum_{i<k} d_i b_i$$

must be surjective. Ideally, it is also injective, but that is not a requirement.

Interesting weights are simple, something like $b_i = B^i$ for some $B \geq 2$.

More Precisely . . .

To define a numeration system $\mathcal{N} = \langle \mathit{\Delta}, \mathrm{D}, \mathrm{val} \rangle$, we need the following pieces.

- A regular language $\mathrm{D} \subseteq \mathit{\Delta}^\star$ that describes the digit strings over $\mathit{\Delta}$ that are admissible representations of numbers.
- A surjective value map $\mathrm{val} : \mathrm{D} \to \mathbb{N}$.

Note that val is **not** required to be injective; in fact, there may be infinitely many representations for the same number.

Representation

As a practical matter, whenever the value map is not injective, it is convenient to have a canonical choice of representation, a representation map $\mathsf{rep} : \mathbb{N} \to \mathsf{D}$ such that $\mathsf{val}(\mathsf{rep}(n)) = n$ for all $n \in \mathbb{N}$.

One natural choice is to pick the length-lex minimal string:

$$\mathsf{rep}(n) = \min_{ll}\big(\, x \in \mathsf{D} \mid \mathsf{val}(x) = n \,\big)$$

If the value map is injective rep is simply the inverse of val.

The Mother of all Numeration Systems

Fix some integer $B \geq 2$. Arguably the easiest choice is

$$\begin{aligned} \Delta &= \{0, 1, \ldots, B{-}1\} \\ \mathsf{D} &= \Delta^\star \\ \mathsf{val}(x) &= \sum_{i<k} x_i B^i \end{aligned}$$

where $x = x_0 x_1 \ldots x_{k-1}$. This is called reverse base B (or reverse radix B), since the LSD comes first in this system.

There are infinitely many representations for any number: $\mathsf{rval}(\varepsilon) = 0$ and $\mathsf{rval}(x0) = \mathsf{rval}(x)$.

Addition is entirely straightforward in this system, it can be carried out by a suitable finite state transducer.

Cleaning Up

As stated, reverse base B is rather too permissive. One often restricts admissible representations.

NE No empty word
ε is not allowed as a representation for $0 \in \mathbb{N}$.

NTZ No trailing zeros
$x \in \mathsf{D}$ implies $x_{-1} \neq 0$.

So for NE and, say, base $B = 2$ we have

$$\mathsf{D} = (\mathsf{0} + \mathsf{1})^+$$

and if we add NTZ we get

$$\mathsf{D} = \mathsf{0} + (\mathsf{0} + \mathsf{1})^* \mathsf{1}$$

Plain Base B

It is customary to start numbers with the MSD, and use a more complicate value map: for $|x| = k$, 0-indexed,

$$\mathsf{val}(x) = \sum_{i<k} x_i B^{k-i-1}$$

This is called base B or radix B notation, we are essentially evaluating x^{op} in reverse base B.

Note that the value map requires knowledge of the length of the string, a feature that often coexists uneasily with finite state machines. Since regular languages and rational relations are closed under reversal, there is no catastrophic difference, but things just tend to get more complicated.

Digression: Weirdness

Weird Digits

Some strange digit sets can be useful on occasion. For example, for base 2, we could alternatively use digits $\{-1, 0, +1\}$, so-called trits, or digits $\{0, 1, 2\}$, so-called hyperbinary digits. E.g., trits can speed up parallel addition.

Weird Weights

Defining weights as powers of B is natural, but we could also use more general recurrences. E.g., Fibonacci numbers can be useful to produce solutions to certain combinatorial problems.

Fibonacci Numeration

There are occasions (combinatorics, coding theory, dynamical systems) where a weird numeration system helps. For example, one could use Fibonacci numbers F_k rather than powers of some base B and digits $\{0, 1\}$:

$$\mathsf{val}(x) = \sum_i x_i F_{i+2}$$

For example, the following strings all have value 32: 111111, 1111001, 1100101, 0010101.

Exercise

Show that any natural number can be written in the Fibonacci system without using two consecutive 1s, so-called Zeckendorf form. Construct a transducer that converts to Zeckendorf form.

Representing Sets of Numbers

Fix some numeration system $\mathcal{N} = \langle \Delta, \mathsf{D}, \mathsf{val} \rangle$.

Suppose $A \subseteq \mathbb{N}$. What does it mean that some plain FSM $\mathcal{A}$ represents A?

Let $L = \mathcal{L}(\mathcal{A}) \subseteq \Delta^\star$. We want

$$L \subseteq \mathsf{D} \quad \text{only admissible strings accepted}$$

$$A = \{ \mathsf{val}(x) \mid u \in L \}$$

So we have a notion of a regular or recognizable set of numbers with respect to numeration system $\mathcal{N}$.

Nitpicking

We could add conditions, say, every vector in A has exactly one representative in L.

If the value map is a bijection, the last condition comes for free. However, if val is not injective one might be tempted to go in the opposite direction:

$$L = \mathsf{val}^{-1}(A) = \bigcup \{\, \mathsf{val}^{-1}(n) \mid n \in A \,\}$$

In this case we will say that $\mathcal{A}$ strongly recognizes A.

Representing Arithmetic

Again, fix a numeration system $\mathcal{N} = \langle \varDelta, \mathsf{D}, \mathsf{val} \rangle$.

Suppose we have an arithmetic function $f : \mathbb{N} \to \mathbb{N}$ (the argument for k-ary functions is entirely similar). What does it mean that some transduction τ represents f?

If $\mathsf{val} : \mathsf{D} \to \mathbb{N}$ is a bijection, the definition is fairly straightforward:

$$\tau \subseteq \mathsf{D} \times \mathsf{D} \text{ total and single-valued}$$
$$\tau(x, y) \Leftrightarrow f\big(\mathsf{val}(x)\big) = \mathsf{val}(y)$$

In general, a reasonable definition appears to be

$$\tau \subseteq \mathsf{D} \times \mathsf{D}$$
$$f = \{\, (\mathsf{val}(x), \mathsf{val}(y)) \mid \tau(x, y), x, y \in \mathsf{D} \,\}$$

Comments

So the transducer only accepts $x{:}y$ when both x and y represent numbers in our system. Also, whenever $f(n) = m$, there are strings $x, y \in \mathsf{D}$ such that $\tau(x, y)$, $\mathsf{val}(x) = n$ and $\mathsf{val}(y) = m$. Lastly, whenever $\tau(x, y)$, we have $f(\mathsf{val}(x)) = \mathsf{val}(y)$.

As in the language case, there is a more restrictive definition: τ strongly represents f iff in addition, for all $x, y \in \mathsf{D}$ such that $\mathsf{val}(x) = n$ and $\mathsf{val}(y) = m$ we already have $\tau(x, y)$.

This may seem more natural than our definition, but it makes the transducers more complicated.

Warmup: Divisibility

Before we tackle rational relations and functions, here is a little finger exercise for regular sets of numbers.

Fix some numeration system $\mathcal{N} = \langle \varDelta, \mathsf{D}, \mathsf{val} \rangle$. For any modulus $m \geq 2$, define the divisibility language as follows:

$$D_{m,\mathcal{N}} = \{\, x \in \mathsf{D} \mid m \mid \mathsf{val}(x) \,\}$$

Since $D_{m,\mathcal{N}}$ is fairly simple, one might expect that for any reasonable numeration system all these divisibility languages are regular.

Base B

It turns out that plain base B is easier to deal with wrto divisibility.

Since we only care about values modulo m it is natural to pick $Q = \{0, 1, \ldots, m{-}1\}$ as state set, $q_0 = 0$, $F = \{0\}$. In radix B we have

$$\mathsf{val}(x\,d) = B\,\mathsf{val}(x) + d \pmod{m}$$

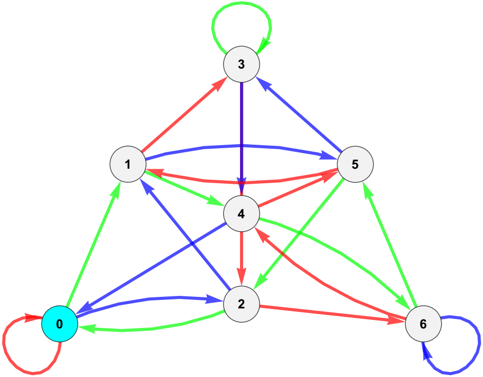
which directly yields the deterministic transition function

$$\delta(p, d) = p\,B + d \bmod m$$

We will call these machines Horner automata, in symbols $\mathcal{H}_{m,B}$.

Careful, though, $\mathcal{H}_{m,B}$ is not minimal in general, just think about $\mathcal{H}_{B,B}$. It's the obvious machine that works, but it typically has redundant states.

Divisibility Base 3, Modulus 7



Automaton $\mathcal{H}_{7,3}$; edge colors: red–0, green–1, blue–2.

Warning: the red edge from 4 to 2 actually starts at 3.

And Reverse Base B?

The infinitely superior LSD first system here causes a bit of a hiccup since

$$\mathsf{val}(xd) = \mathsf{val}(x) + d\,B^{|x|} \pmod{m}$$

so the machine needs to keep track of the numbers $B^{|x|} \bmod m$.

The sequence $(B^k \bmod m)_{k\geq 0}$, depends very much on divisibility properties of m and B. Without worrying about the details, it has to be ultimately periodic: there are finite sequences α and β over $\mathbb{N}$ such that $(B^k \bmod m)_{k\geq 0} = \alpha\,\beta^\omega$. In fact, $|\alpha| + |\beta| \leq m$. We call $\alpha;\beta$ the fundamental sequence of B and m.

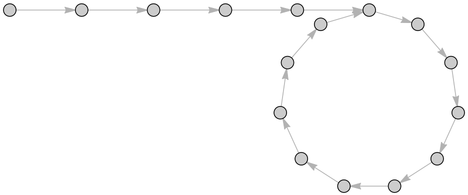
We can wrap this up in a fairly nice algebraic way.

Transient Mod

Here is a function that describes the position of a particle moving on a lasso with transient t and period p.

$$\mathsf{rem}_{t,p}(i) = \begin{cases} i & \text{if } i < t, \\ t + (i - t) \bmod p & \text{otherwise.} \end{cases}$$

Write $\mathsf{succ}(i) = \mathsf{rem}_{t,p}(i+1)$ and write $\mathbb{N}_{t,p}$ for the structure $\langle \{0, 1, \ldots, t{+}p{-}1\}, \mathsf{succ} \rangle$. A picture of $\mathbb{N}_{5,11}$:



Reverse Divisibility Automaton

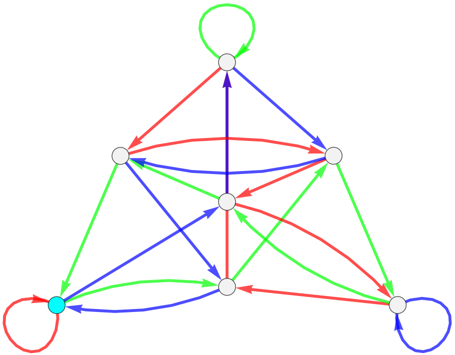
To build the divisibility machine for reverse base B modulo m, let $\alpha;\beta$ be the fundamental sequence, $t = |\alpha|$ and $p = |\beta|$.

We can now build a divisibility automaton $\mathcal{H}^r_{m,B}$ as follows:

$$\begin{aligned} Q &= \{0, \ldots, m{-}1\} \times \mathbb{N}_{t,p} \\ \delta((p,s),d) &= \bigl(p + d\,\beta_s \bmod m, \mathsf{succ}(s)\bigr) \\ q_0 &= (0,0) \\ F &= \{\, (0,s) \mid s \in \mathbb{N}_{t,p} \,\} \end{aligned}$$

Note that this machine has size $O(m^2)$, though the accessible part can be smaller and the minimal machine can be smaller yet.

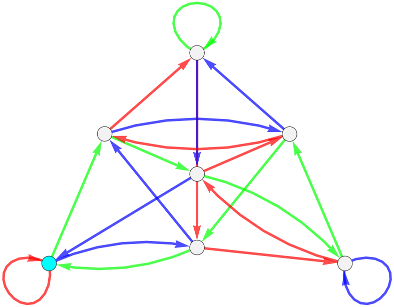
Divisibility Reverse Base 3, Modulus 7

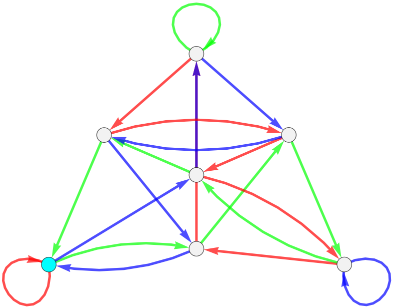


$\mathcal{H}^r_{7,3}$ has $42 = 7 \cdot 6$ states

fundamental sequence $;1, 3, 2, 6, 4, 5$ (transient 0, period 6).

Comparison





Plain base 3 on the left, reverse base 3 on the right.

Both machines are minimal.

But Why?

From the picture, $\min \mathcal{H}^r_{7,3}$ is the reversal of $\mathcal{H}_{7,3}$.

To see why, note that $\mathcal{H}_{7,3}$ is a permutation automaton: all the transition functions δ_a are permutations of the state set.

By Brzozowski, that means that $\mathsf{pow}\big(\mathcal{H}^{\mathsf{op}}_{7,3}\big)$ is the minimal automaton for reverse base 3. But $\mathcal{H}^{\mathsf{op}}_{7,3}$ is a DFA, done.

Careful, though, it is not true in general that $\mathcal{H}_{m,B}$ is a permutation automaton.

1 Numeration Systems

2 Presburger Arithmetic

3 Automatic Structures

4 Deciding Presburger Arithmetic

A Verification Problem

Suppose you are implementing a dynamic programming algorithm that has to fill out an $n \times n$ array A. The algorithm

- initializes the first row and the first column,
- then fills in the whole array according to

$$A[i,j] = F(A[i-1,j], A[i,j-1])$$

- lastly, reads off the result in $A[n-1,n-1]$.

We would like to check that all the array access operations are safe, and that the result is properly computed.

And, we want to do so automatically.

Filling In

One of the problems is that the recurrence can be implemented in several ways:

```
// column by column
for i = 1 .. n-1 do
for j = 1 .. n-1 do
      A[i,j] = F( A[i-1,j], A[i,j-1] )


// row by row
for j = 1 .. n-1 do
for i = 1 .. n-1 do
      A[i,j] = F( A[i-1,j], A[i,j-1] )


// by diagonal
for d = 1 .. 2n-3 do
for i = 1 .. d do
      A[i,d-i+1] = F( A[i-1,d-i+1], A[i,d-i] )
```

Correctness

For a human, it is easy to see that the row-by-row and column-by-column methods are correct. The diagonal approach already requires a bit of thought: why $2n - 3$?

Automatic verification requires a some amount of arithmetic, so there is an immediate difficulty: full arithmetic is undecidable, in fact even Σ_1 statements are already undecidable.

The problem is that the terms in the language of arithmetic are essentially multivariate polynomials and Matiyasevic has shown that finding integer roots of those is hard.

Question:
Can we get away with some decidable "baby arithmetic"?

Presburger Arithmetic

Here is a radical proposal: let's just forget about multiplication. More precisely, we restrict ourselves to the structure

$$\mathfrak{N}_{-} = \langle \mathbb{N}, +, 0, 1; < \rangle$$

As far as first-order logic is concerned, this structure is actually a bit overly verbose, we could get away with just $\langle \mathbb{N}, + \rangle$. To see why, note that we can define all the rest:

$$\begin{array}{ll}
x = 0 & \forall\, z\, (x + z = z) \\
x \leq y & \exists\, z\, (x + z = y) \\
x < y & x \leq y \land x \neq y \\
x = 1 & 0 < x \land \forall\, z\, (0 < z \Rightarrow x \leq z) \\
x = y + 1 & y < x \land \forall\, z\, (y < z \Rightarrow x \leq z)
\end{array}$$

Terms

The terms in this language are pretty weak, no more than affine combinations:

$$t(\boldsymbol{x}) = c + \sum c_i x_i$$

where all the c, c_i are constant, $c, c_i \in \mathbb{N}$.

This is easy to check by induction on terms.

In particular, we have lost multivariate polynomials–which is a good thing, since otherwise Matiyasevic's theorem would automatically doom any attempt at finding a decision algorithm.

Semilinear Sets

Definition

A set $A \subseteq \mathbb{N}$ is linear if $A = \{\, c + \sum c_i x_i \mid x_i \geq 0 \,\}$.
Here c is the constant and the c_i are the periods.

A set $A \subseteq \mathbb{N}$ is semilinear if it is the finite union of linear sets.

Actually, this is just the one-dimensional case; one can generalize easily to subsets of $\mathbb{N}^d$. Note that every finite set is semilinear. Intuitively, the semilinear sets are the ultimately periodic subsets of $\mathbb{N}$.

Lemma

Semilinear sets are closed under union, intersection and complement.

Definability

Clearly, semilinear sets are definable in Presburger Arithmetic, using no more than Σ_1 formulae:

$$z \in A \iff \exists\, \boldsymbol{x}\, \big(t_1(\boldsymbol{x}) = z \lor t_2(\boldsymbol{x}) = z \lor \ldots \lor t_k(\boldsymbol{x}) = z\big)$$

Theorem

The sets definable in Presburger Arithmetic are exactly the semilinear sets.

This result is somewhat surprising: your intuition might tell you that more complicated quantifier structures would produce more complicated sets.

Digression: Tally Languages

Suppose we code natural numbers in unary: $n \mapsto a^n$.

Then every set $A \subseteq \mathbb{N}$ corresponds to a tally language $L_A \subseteq \{a\}^\star$.

Theorem

A set $A \subseteq \mathbb{N}$ is semilinear if, and only if, L_A is regular.

With a little bit of automata theory lying around, this explains everything one wants to know about semilinear sets.

Digression: Presburger's Approach

Presburger used an important method called quantifier elimination: by transforming a formula into another, equivalent one that has one fewer quantifiers. Ultimately, we can get rid of all quantifiers.

The remaining quantifier-free formula is equivalent to the original one, and is easily tested for validity. A single step takes us from

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

to an equivalent formula

$$\Phi \equiv \Phi' = \exists\, x_1\, \forall\, x_2\, \exists\, x_3\, \ldots \varphi'(\boldsymbol{x})$$

where the elimination variable z is no longer present.

Universal quantifiers are handled via $\forall \equiv \neg\exists\neg$.

How Bad Can It Be?

Full multiplication is absent, but multiplication by a **constant** is available; for example

$$y = 3 * x \iff y = x + x + x$$

We can also do modular arithmetic with **fixed modulus**:

$$\begin{aligned} y = x \bmod 3 &\iff \exists\, z\, (x = 3 * z + y \wedge y < 3) \\ y = x \operatorname{div} 3 &\iff \exists\, z\, (x = 3 * y + z \wedge z < 3) \end{aligned}$$

This may seem pretty feeble and would probably lead Wurzelbrunft to conclude that it's trivial to come up with a decision algorithm.

Poor Wurzelbrunft.

A Bad Formula

Here is a fairly simple formula in Presburger arithmetic:

$$\Phi \equiv \exists\, u\, \forall\, v\, \big(u < v \Rightarrow \exists\, x, y\, (3x + 5y = v)\big)$$

How would a decision algorithm tackle this formula?

A human being having suffered through 128/151 knows what the answer is: true. But the reason is uncomfortably complicated, we need to know that 3 and 5 are coprime, a concept that is firmly rooted in the theory of multiplication.

Really Bad

Worse, the problem naturally generalizes to several multipliers. One wants to compute the largest number v such that

$$v = \boldsymbol{a} \circ \boldsymbol{x} = a_1 x_1 + a_2 x_2 + \ldots + a_d x_d$$

has no solution $\boldsymbol{x} \in \mathbb{N}^d$.

This is often written as the Frobenius function $g(a_1, \ldots, a_d)$.

We can easily define the Frobenius function in Presburger arithmetic:

$$g(\boldsymbol{a}) = b \iff \neg \exists\, \boldsymbol{x}\, (b = \boldsymbol{a} \circ \boldsymbol{x}) \wedge \forall\, z > b\, \exists\, \boldsymbol{x}\, (z = \boldsymbol{a} \circ \boldsymbol{x})$$

So What?

The only easy proposition about the Frobenius function is the 2-dimensional case:

$$g(a_1, a_2) = a_1 a_2 - a_1 - a_2$$

The general case is notoriously hard. E.g., it is $\mathbb{NP}$-hard to compute $g(\boldsymbol{a})$. There is a polynomial time algorithm when d is fixed, though.

Since a general decision algorithm would provide at least some sort of insight into to the Frobenius function, there is no hope that any such algorithm could be simple.

1 Numeration Systems

2 Presburger Arithmetic

3 **Automatic Structures**

4 Deciding Presburger Arithmetic

Automatic Structures

Suppose we have some mathematical structure $\mathcal{X} = \langle X; R \rangle$ where R is a binary relation on X. Generalization to multiple relations/functions are obvious.

Definition

$\mathcal{X}$ is automatic if there is a regular language $\mathsf{Nm} \subseteq \Sigma^\star$ and a surjective naming function $\nu : \mathsf{Nm} \to X$ such that

1. the binary relation on Nm, $\nu(u) = \nu(v)$, is synchronous,
2. the binary relation on Nm, $R\big(\nu(u), \nu(v)\big)$, is synchronous.

Nm is the set of names for the actual elements in $\mathcal{X}$.

Comments

One does not require the naming function to be injective, the same object in X may have several (even infinitely many) names. If ν is a bijective, then the first condition is trivially satisfied.

There is no condition on ν being computable in some simple manner, it just has to be defined on a regular set of words and be compatible with $=$ and R as in the definition.

In many concrete cases there is a "natural" choice for ν and Nm, but that is not part of the definition of automaticity.

Strong Representation

This definition of automaticity is the correct one in the sense that it produces the desired result.

Note that we require our FSM representation to be fairly strong: we insist on a synchronous automaton $\mathcal{A}_R$ such that for any $x, y \in X$ we must have have

$$R(x,y) \iff \forall\, u \in \nu^{-1}(x), v \in \nu^{-1}(y) \;\big(\mathcal{A}_R(u,v)\downarrow\big)$$

In other words, any name works, we don't need to pick a nice one for the automaton to produce the right answer.

A Gap

The idea of automaticity is half a century old.

Bernard R. Hodgson

Théories décidables par automate fini

Ph.D. thesis, 1976, Université de Montréal

Sadly, no one payed any attention until it was reinvented 20 years later.

A broad study of automaticity really started taking off around 2000.

Hodgson already had a number of interesting examples: dense and discrete linear orders, Presburger arithmetic, p-adic numbers.

He showed that automatic structures have certain closure properties wrto product constructions.

And, he realized that one can use the same approach based on automata operating on infinite words (ω-automata).

Wordprocessing in Groups

Clever title of a book published in 1992 by D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Patterson, and W. P. Thurston.

This approach uses the Cayley graph of a group rather than the usual first-order structure and there are subtle differences between the two methods.

They provide a quadratic time algorithm based on finite state machines that solves the word problem for certain groups that are important in low-dimensional topology.

1. Numeration Systems
2. Presburger Arithmetic
3. Automatic Structures
4. **Deciding Presburger Arithmetic**

Naming Numbers

We want to think of

$$\mathfrak{N}_+ = \langle \mathbb{N}; + \rangle$$

as an automatic structure.

So, we need a naming map $\nu : \mathsf{Nm} \to \mathbb{N}$ where $\mathsf{Nm} \subseteq \Sigma^\star$ is some regular language.

No problem, any of the standard numeration systems will work. And, it does not matter which one we choose as far as decidability results are concerned.

It does matter, though, for the actual algorithm, the machines will be somewhat different.

Reasonable Convention

We will adopt the following conventions for the representation of $\mathbb{N}$ in $\mathbf{2}^\star$.

- reverse binary (LSD first)
- trailing zeros are allowed (TZ)
- $0 \in \mathbb{N}$ is denoted by $\mathtt{0}$, not the empty word (NE)

In this case, $\mathsf{Nm} = (\mathtt{0} + \mathtt{1})^+$. Since we allow trailing zeros, we can avoid padding and work over $\mathbf{2}$ (more precisely, we pad with zero). The naming map is typically called the value map here, it associates a digit string with a numerical value:

$$\mathsf{val} : \mathbf{2}^+ \longrightarrow \mathbb{N}$$

This map is most natural, but infinite-to-one.

Atomic Formulae

At any rate, in our case, there are only two possible atomic formulae:

- $x = y$
- $x + y = z$

We could add $x < y$ to improve the expressiveness of the language a bit.

So we have 3 synchronous transducers $\mathcal{A}_{=}^{(2)}$, $\mathcal{A}_{<}^{(2)}$ and $\mathcal{A}_{+}^{(3)}$ (2, 2, and 3 tracks, respectively) that test these predicates, given arbitrary names for the natural numbers in question.

Clearly we could also handle the constants 0 and 1 (and arbitrary constants for that matter).

Efficiency

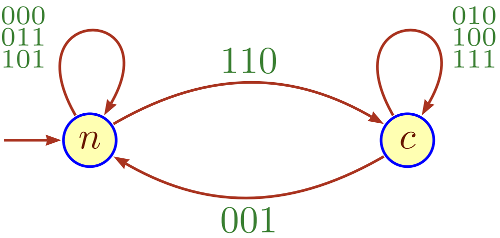
Again, we don't have to include $\mathcal{A}_{<}^{(2)}$ in our list of basic machines, we could replace every occurrence of an atomic formula $s < t$ by the formula

$$\forall\, d\left(s + d = t \wedge s \neq t\right)$$

The decision algorithm would build a FSM that handles this condition, but it is better to lovingly handcraft an optimized machine once and for all.

More generally, one probably would like to have a little library of optimized machines for phrases that appear often.

Raw Addition



This is the core of the transducer for addition, with states “no carry” and “carry,” using reverse binary as the numeration system.

Alas, this transducer ignores all other aspects of a numeration system (empty-word, trailing-zeros).

Warning

The machines on the next few slides do not use a padding symbols; instead we cheat by padding with 0s. They are correct in spirit, but actually wrong as far as our definitions go.

To wit, they require the right number of trailing zeros in the names chosen to represent the natural numbers in question. That violates the definition of automaticity.

Exercise

Figure out how to fix this problem by using a padding symbol.

Who Cares?

Suppose we want to express relational composition $\exists\, y\, (R(x,y) \wedge R(y,z))$. Suppose we have corresponding names $u, v, w \in \mathsf{Nm}$.

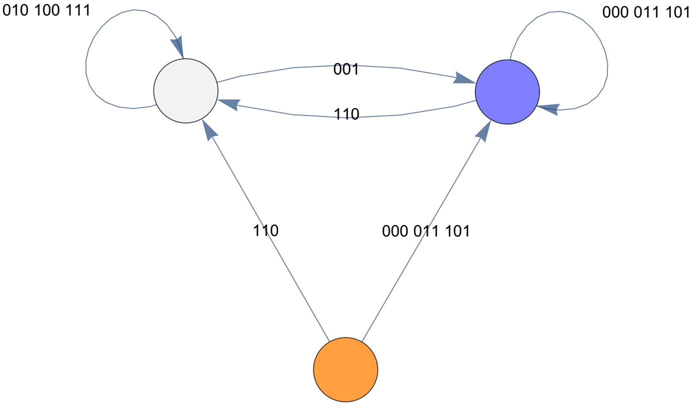
With our type of machine it may happen that

$$\mathcal{A}_R \text{ accepts } u{:}v \text{ and } v\mathsf{0}^{42}{:}w$$

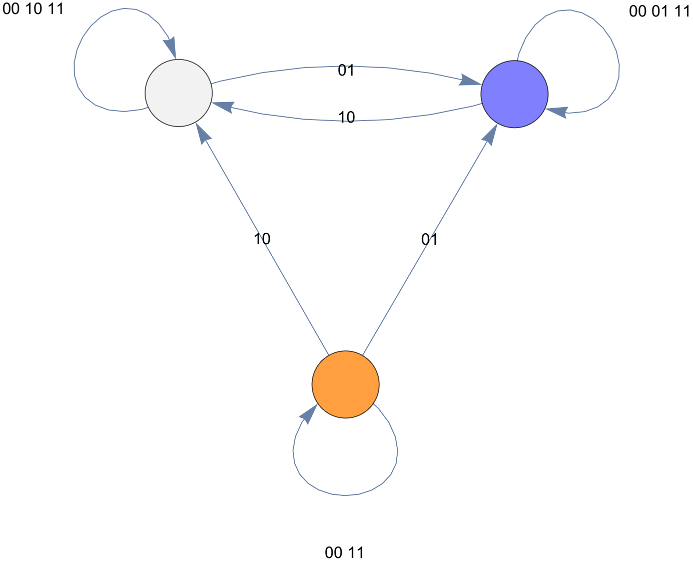
But the 3-track product machine does not accept $u{:}v{:}w$.

The construction fails since we cannot cope with 2 different representations for the witness.

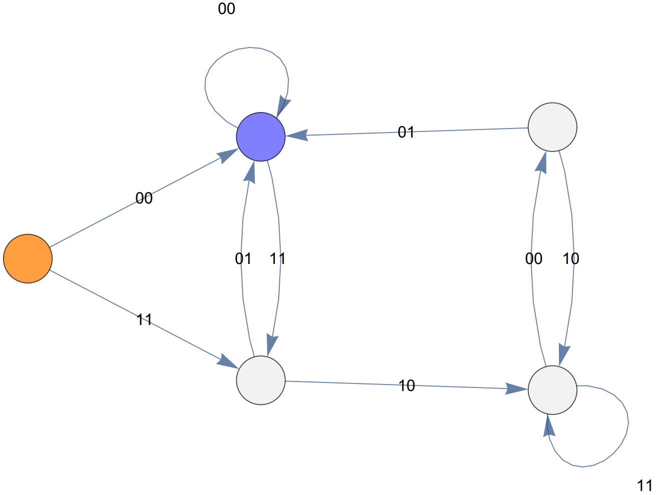
Addition NE, TZ



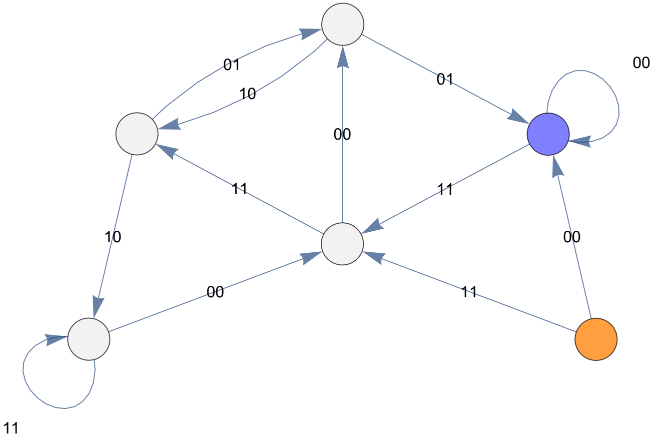
Less-Than



Multiply by 3



Multiply by 5



A Sentence

Consider the sentence

$$\Phi \equiv \exists\, x\, \forall\, y \left(x < y \Rightarrow \exists\, u, v\, (3 * u + 5 * v = y)\right)$$

Thanks to our brilliant choice of coefficients, Φ is actually true.

We use a 5-track machine $\mathcal{A}$ with the intended meaning:

u	existential quantifier
v	existential quantifier
u'	$u' = 3u$
v'	$v' = 5v$
z	$z = u' + v'$

More Precisely

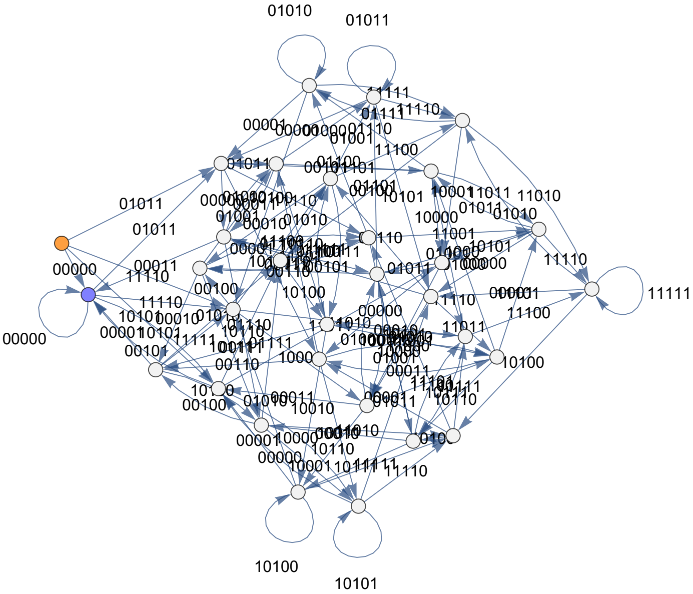
We combine a multiply-by-3 and a multiply-by-5 machine with an adder:

$$\mathcal{A} = \mathsf{emb}^{(5)}_{1,3}\big(\mathsf{mult}_3\big) \times \mathsf{emb}^{(5)}_{2,4}\big(\mathsf{mult}_5\big) \times \mathsf{emb}^{(5)}_{3,4,5}\big(\mathcal{A}_+\big)$$

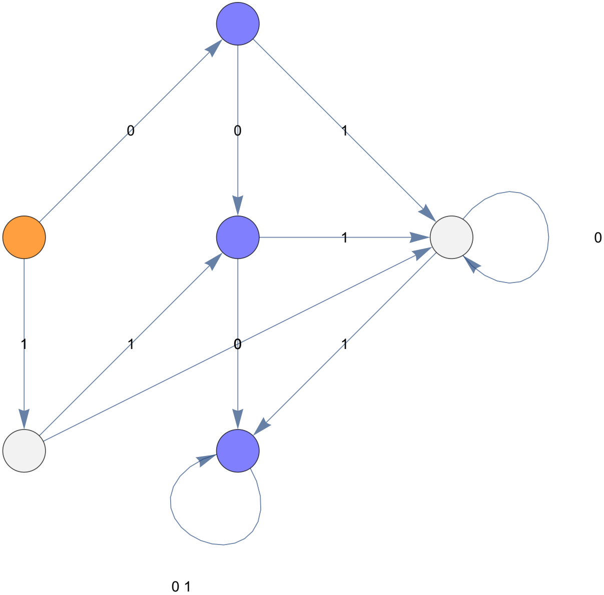
Projecting away all but the z-track

$$\mathcal{B} = \mathsf{prj}^{(5)}_{1,2,3,4}(\mathcal{A})$$

produces a plain FSM that recognizes all linear combinations of 3 and 5.


01010
01011
11111
11110
00001
01111
01001
11100
01011
01011
00000
11110
00011
00100
11110
10101
00010
00001
10101
00101
11111
00000
11010
01010
10000
11001
10101
01011
11110
00000
11100
11011
10100
00011
10010
00100
00001
10000
10110
00000
10001
11110
10100
10101

Minimized $\mathcal{B}$



Onward

It now suffices to check that

$$\Phi \equiv \exists\, x \,\forall\, y \,\big(x < y \Rightarrow \mathcal{B}(y)\big)$$

The universal quantifier needs to be rewritten:

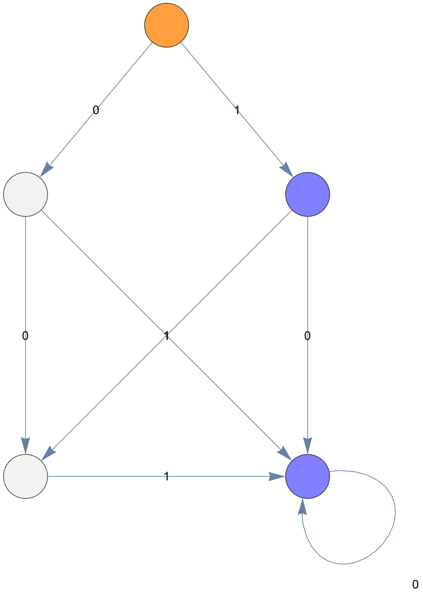
$$\Phi \equiv \exists\, x \,\neg \,\exists\, y \,\big(x < y \land \neg\mathcal{B}(y)\big)$$

The $\exists\, y \,\big(x < y \land \neg\mathcal{B}(y)\big)$ part is handled by more embeddings:

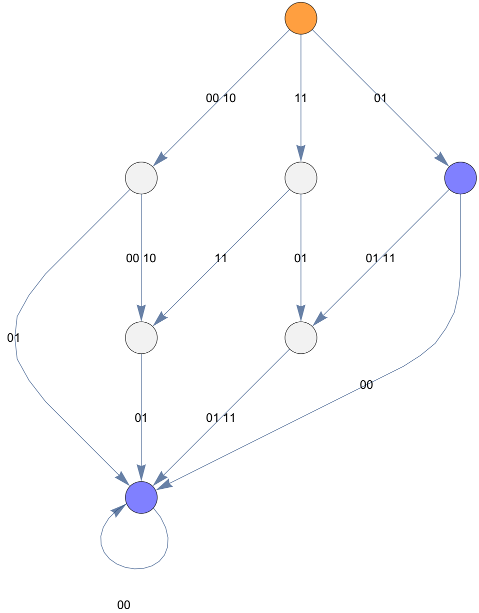
$$\mathcal{C} = \mathsf{emb}^{(2)}_{1,2}\big(\mathsf{less}\big) \times \mathsf{emb}^{(2)}_{2}\big(\neg\mathcal{B}\big)$$

For the complement we have to deal with the numeration system; only strings in Nm are allowed.

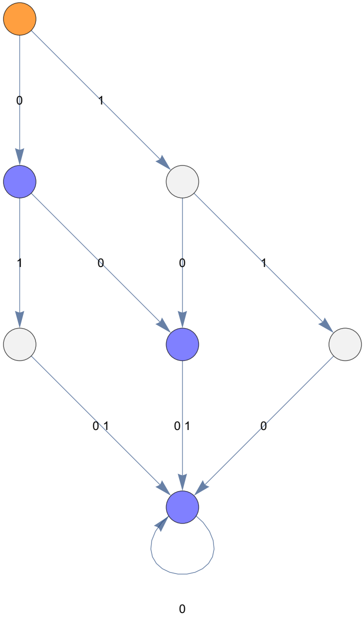
Span Complement



$\neg\mathcal{B}$ recognizes $1, 2, 5, 6.$


00 10
11
01
01
00 10
11
01
01 11
01
00
01
01 11
00

Projecting $\mathcal{C}$



least number not accepted: 7

hence witness $x = 7$ works

Isomorphism Problem

Since automatic presentations depend on a naming function $\nu : \mathsf{Nm} \to X$ it is far from clear whether two automatic presentations describe the same underlying first-order structure. This is known as the isomorphism problem.

Theorem

The isomorphism problem for automatic structures is undecidable.

In fact, the problem is outside of the arithmetical hierarchy and belongs to the analytical hierarchy, at level Σ^1_1.

Non-Automaticity

By the same token, it is fairly difficult to make sure that a given structure $\mathcal{X}$ fails to be automatic. Here are some examples of non-automatic structures:

- Additive rational numbers $\langle \mathbb{Q}; + \rangle$
- Divisibility of naturals $\langle \mathbb{N}; | \rangle$
- Skolem arithmetic $\langle \mathbb{N}; \cdot \rangle$
- Free semigroup $\{a, b\}^\star$
- Structures with a pairing function

Interpretations

Let $\mathcal{A}$ and $\mathcal{B}$ be two relational, first-order structures with signatures $\tau(\mathcal{A})$ and $\tau(\mathcal{B})$. The signatures may well be different, so their first-order theories use a different vocabulary.

For simplicity, assume that $\mathcal{A}$ has just one binary relation R.

Definition

An n-dimensional interpretation of $\mathcal{A}$ in $\mathcal{B}$ consists of first-order formulae in the language of $\mathcal{B}$

$$\delta(\boldsymbol{x}) \qquad \phi_R(\boldsymbol{x}, \boldsymbol{y})$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are n-vectors of distinct variables. Furthermore, there is a surjective coordinate map $f : \delta^{\mathcal{B}} \to A$ such that

$$\mathcal{A} \models R(f\boldsymbol{b}_1, f\boldsymbol{b}_2) \iff \mathcal{B} \models \phi_R(\boldsymbol{b}_1, \boldsymbol{b}_2)$$

Extension to Formulae

We can associate any formula Φ in the vocabulary of $\mathcal{A}$ with an interpretation $\Phi^{\mathcal{I}}$, a formula in the vocabulary of $\mathcal{B}$ defined as follows:

- Replace any atomic formula $R(u, v)$ by $\phi_R(\boldsymbol{x}, \boldsymbol{y})$.
- Keep the propositional part in tact.
- Relativize every quantifier to δ:

$$\forall\, u\, \phi(u) \rightsquigarrow \forall\, \boldsymbol{x}\, \big(\delta(\boldsymbol{x}) \Rightarrow \phi(\boldsymbol{x})\big)$$
$$\exists\, u\, \phi(u) \rightsquigarrow \exists\, \boldsymbol{x}\, \big(\delta(\boldsymbol{x}) \land \phi(\boldsymbol{x})\big)$$

Interpretation Lemma

Theorem

Let $\mathcal{B}$ interpret $\mathcal{A}$ with coordinate map $\mathcal{I}$. Then for any first-order formula $\Phi(x)$ in the vocabulary of $\mathcal{A}$ and $b \in \mathcal{B}$:

$$\mathcal{A} \models \Phi(f\boldsymbol{b}) \iff \mathcal{B} \models \Phi^{\mathcal{I}}(\boldsymbol{b})$$

In particular for a sentence we have

$$\mathcal{A} \models \Phi \iff \mathcal{B} \models \Phi^{\mathcal{I}}$$

So all first-order questions about $\mathcal{A}$ can be answered in $\mathcal{B}$.

In particular, if we have a decision algorithm for $\mathcal{B}$ we can also apply it to $\mathcal{A}$: just check $\Phi^{\mathcal{I}}$ instead of Φ.

Divisibility

Here is an extension of Presburger arithmetic based on the special divisibility relation for $m \geq 2$:

$$x \mid_m y \iff x \mid y \wedge \exists\, k\, (x = m^k)$$

$$\mathfrak{N}_m = \langle \mathbb{N}; +, \mid_m \rangle$$

$\mathfrak{N}_m$ is still automatic: we can write numbers in reverse base m.

Addition is synchronous as always. The divisibility relation is also easily synchronous.

Trees

Write $\mathit{\Delta}_m = \{0, 1, \ldots, m{-}1\}$ for the usual digit alphabet in base m.

We can think of all words over $\mathit{\Delta}_m$ as an m-ary tree. Let $\sigma_d(x) = xd$ be the dth child of x in that tree, write $\prec$ for the prefix partial order on ${\mathit{\Delta}_m}^\star$ and write $\mathsf{el}(x, y)$ if x and y have the same length.

The tree structure of order m is defined by

$$\mathsf{tree}_m = \langle {\mathit{\Delta}_m}^\star; \sigma_0, \ldots, \sigma_{m-1}, \prec, \mathsf{el} \rangle$$

Interpretation

Theorem

The structures $\mathfrak{N}_m$ and tree_m are mutually interpretable in each other.

Sketch of proof. To interpret, say, tree_m in $\mathfrak{N}_m$, associate a word $u = u_0 u_1 \ldots u_{k-1}$ in ${\Delta_m}^\star$ (i.e., a branch in the tree) with the numerical value

$$m^k + \sum_i u_i m^i$$

The m^k forces the MSD to be 1 and we have to choose δ accordingly. First note that for all x there is exactly one y such that

$$\alpha(x, x') \equiv x' \mid_m x' \wedge x' \leq x < mx'$$

Hence we can set

$$\delta(x) \equiv \exists\, x' \left(\alpha(x, x') \wedge x < 2x'\right)$$

Using α, we can express $\sigma_d(u) = v$ as

$$\psi_d(x, y) \equiv \exists\, x' \left(\alpha(x, x') \wedge y = x + (m + d - 1)x'\right)$$

The prefix relation translates into

$$\pi(x, y) \equiv \exists\, x', z \left(\alpha(x, x') \wedge x' \mid_m z \wedge y = x + z\right)$$

Lastly, equal length/equal level translates into

$$\lambda(x, y) \equiv \exists\, x', y' \left(\alpha(x, x') \wedge \alpha(y, y') \wedge x' = y'\right)$$

Done. □

Universal Automatic Structures

Here is a rather surprising result that states that, in the sense of interpretability, the structures $\mathfrak{N}_m$ and tree_m are the most complicated automatic structures.

Theorem

A structure is automatic iff it can be interpreted in $\mathfrak{N}_m$ iff it can be interpreted in tree_m*.*

So, as a matter of principle, we could solve all first-order queries about automatic structures by a translation to a carefully optimized decision algorithm for $\mathfrak{N}_m$ and/or tree_m.