CDM

Presburger Arithmetic

K. Sutner

Carnegie Mellon University

Fall 2025

Our next project is to show how automaticity can be used to build a decision algorithm for first-order logic of a fragment of arithmetic (addition only, no multiplication).

Warning:

There is a major difference between this and our two previous examples:

- invertible Mealy automata
- elementary cellular automata

In both cases the objects we are working with are naturally binary strings.

For arithmetic we have to deal with integers.

Arithmetic

We will work over the structure

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

Essentially arithmetic without multiplication.

To handle the carrier set $\mathbb{N}$, we have to encode natural numbers as strings.

In other words, we need to fix a suitable notation system for natural numbers that that our machines can work on.

And we need to express operations and relations in terms of our representation.

Numeration Systems

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

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 $d = d_0 d_1 \dots d_{k-1}$ is determined by weights $b_i \in \mathbb{N}_+$, $i \geq 0$, as follows. The value map $\mathsf{val} : \Delta^\star \to \mathbb{N}$

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

must be surjective: every number must have a name.

We only care about the standard weights $b_i = B^i$ for some base $B \geq 2$.

More Precisely . . .

A numeration system $\mathcal{N} = \langle \Delta, \mathsf{D}, \mathrm{val} \rangle$ consist of the following.

- A digit alphabet Δ.
- A recognizable language $\mathsf{D} \subseteq \Delta^\star$.
- A surjective value map $\mathrm{val} : \mathsf{D} \to \mathbb{N}$.

D consists of all admissible representations of numbers.

The value map is **not** required to be injective; the system may have redundant ways to express numbers.

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: $\mathsf{val}(\varepsilon) = 0$ and $\mathsf{val}(x0) = \mathsf{val}(x)$.

Addition is entirely straightforward in this system, a standard ripple-carry-adder translates into a synchronous transducer.

Cleaning Up

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

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

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

In a NEW numeration system with, say, base $B = 2$ we have

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

and if we add NTZ we get

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

Plain Base B

Alas, many misguided people prefer to start with the MSD, and use a more complicated 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.

This value map requires knowledge of the length of the string, a feature that 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.

Representing Sets of Numbers

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

Suppose $A \subseteq \mathbb{N}$ is some arithmetic set.
How do we represent A by some plain FSM $\mathcal{A}$?

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

$$L \subseteq \mathsf{D}$$
$$A = \{\, \mathsf{val}(x) \mid x \in L \,\}$$

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

Redundancy

If $\mathcal{N}$ has exactly one string for each number it is called unambiguous or non-redundant. In this case, the value map is a bijection.

In a redundant system it is natural to insist that every name work:

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

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

Otherwise things could go sideways: think about $A \cap B$.

Representing Operations

For simplicity, suppose we have a binary arithmetic function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

What does it mean that some transduction T_f represents f?

Clearly we need $T_f \subseteq \mathsf{D} \times \mathsf{D} \times \mathsf{D}$. Moreover

$$T_f(x, y, z) \iff f\big(\mathsf{val}(x), \mathsf{val}(y)\big) = \mathsf{val}(z)$$

If $\mathcal{N}$ is non-redundant, then T_f is actually a function on $\mathsf{D} \times \mathsf{D}$.

Avoiding Phantoms

When we manipulate our transducers we need to maintain the condition that each track must be a word in D.

This is easy for all operations except for complement: say, we have a machine $\mathcal{A}$ recognizing some set $A \subseteq \mathbb{N}$.

To get a machine $\mathcal{A}'$ for $\mathbb{N} - A$ we need to

- determinize the machine
- flip final/non-final states
- intersect with D.

Without the last step, $\mathcal{A}'$ may accept "phantom strings" $x \notin$ D so it looks like $\mathbb{N} - A \neq \emptyset$ when in fact $A = \mathbb{N}$.

Suppose we have some mathematical structure $\mathcal{X} = \langle X; R \rangle$ where R is a k-ary relation on X.

Definition

$\mathcal{X}$ is automatic if there is a regular language Nm $\subseteq \Sigma^\star$ of names and a surjective value function ν : Nm $\to X$ such that

1. the binary relation on Nm, $\nu(u) = \nu(v)$, is synchronous,
2. the k-ary relation on Nm, $R\big(\nu(u_1), \ldots, \nu(u_k)\big)$, is synchronous.

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

Comments

A plain finite state machine can check whether a string denotes some object in $\mathcal{X}$.

The naming system may be redundant, but we can synchronously check whether two names denote the same object.

The actual relations in $\mathcal{X}$ translate into synchronous relations on Nm.

Ominous Comment

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

To be sure, in many concrete cases there is a canonical choice for ν and Nm. This is true in particular in arithmetic where we simply resort to standard numeration systems.

But: neither computability nor canonicity is part of the definition of automaticity.

Recall: Wordprocessing in Groups

The 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.

The authors essentially handle groups whose Cayley graphs are automatic. This is different from the automaton groups we looked at earlier.

They develop a quadratic time algorithm that solves the word problem for certain groups that are important in low-dimensional topology.

Fragments of Arithmetic

Full arithmetic is notoriously undecidable, but how about a fragment like

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

which has no multiplication. In practice one often uses $\langle \mathbb{N}; +, 0, 1, < \rangle$, the constants and order are definable in terms of just addition.

Since there is no multiplication there are no polynomials and all the standard undecidability results fall by the wayside.

Maybe we could decide first-order arithmetic over $\mathfrak{N}_+$?

Warning

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:

$$y = x \bmod 2 \iff \exists\, z\, (x = 2 * z + y \wedge y < 2)$$
$$y = x \operatorname{div} 2 \iff \exists\, z\, (x = 2 * y + z \wedge z < 2)$$

A slightly non-trivial example of a Presburger formula:

$$\exists\, x \forall\, y \exists\, u, v\, (x < y \Rightarrow y = 5 * u + 7 * v)$$

Is it valid?

Presburger Arithmetic is Decidable

Without multiplication, arithmetic is much less complicated.

Theorem (M. Presburger 1929)

First-order logic over $\mathfrak{N}_+$ is decidable.

This result seemed like a major boost to Hilbert's program: first-order logic is sound and complete, and it can handle an interesting fragment of arithmetic.

Of course, what was really needed is a similar result for all of arithmetic. Alas ...

Full Story

In 1929, Presburger showed that arithmetic without multiplication (Presburger arithmetic) is decidable.

In 1930, Skolem proved that arithmetic without addition (Skolem arithmetic) is decidable.

In 1931, Gödel showed that full Peano arithmetic is incomplete.

Followed by Church and Turing who showed in 1936 that arithmetic is undecidable.
Lastly, in 1970, Matiyasevich showed that even checking for integer roots of integer polynomials is undecidable.

Deciding Presburger Arithmetic

There are at least three ways to tackle this problem.

- Quantifier elimination
- Monadic second-order logic and ω-automata
- Automaticity and ordinary finite state machines

Presburger's original algorithm is based on quantifier elimination and purely syntactic in nature.

Büchi developed the MSO approach in the 1960s.

Automaticity essentially dates back to Nerode in the 1990s.

A Gap

Actually, 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 paid attention until Nerode reinvented it 20 years later.

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

It Works

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).

Efficiency?

Unfortunately, it turns out that the computational complexity of Presburger arithmetic is pretty bad:

$$\Omega(2^{2^{cn}}) \qquad \text{and} \qquad O(2^{2^{2^{cn}}})$$

Meaning: every algorithm for Presburger is at least doubly exponential on some inputs, and there is a triple exponential algorithm.

In general, all the algorithms need tender care and feeding. And, they can handle only limited instances.

Naming Numbers

We want to think of

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

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, we can use our standard numeration system $\mathcal{N}$:

reverse binary, no empty word, no trailing zeros.

Hence $\mathsf{Nm} = \mathsf{D} = 0 + (0 + 1)^* 1$ and our system is unambiguous.

Atomic Formulae

In our case, there are three atomic formulae:

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

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.

In practice we would also implement the constants 0 and 1 (or other small constants).

Efficiency

The order relation $s < t$ is first-order definable in terms of addition:

$$\exists\, d \left(s + d = t \wedge d \neq 0 \right)$$

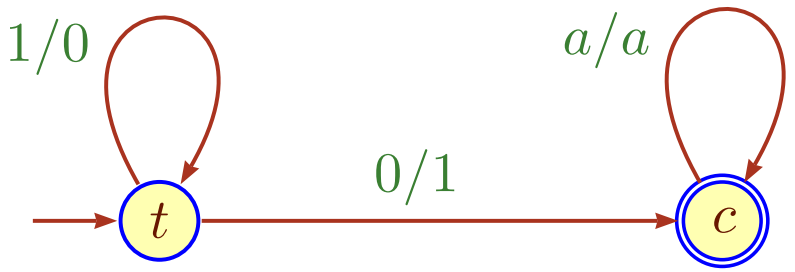
but it is better for efficiency reasons to add it as a primitive.

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, in practice one would construct a little library of optimized machines for phrases that appear often.

Vanilla Successor

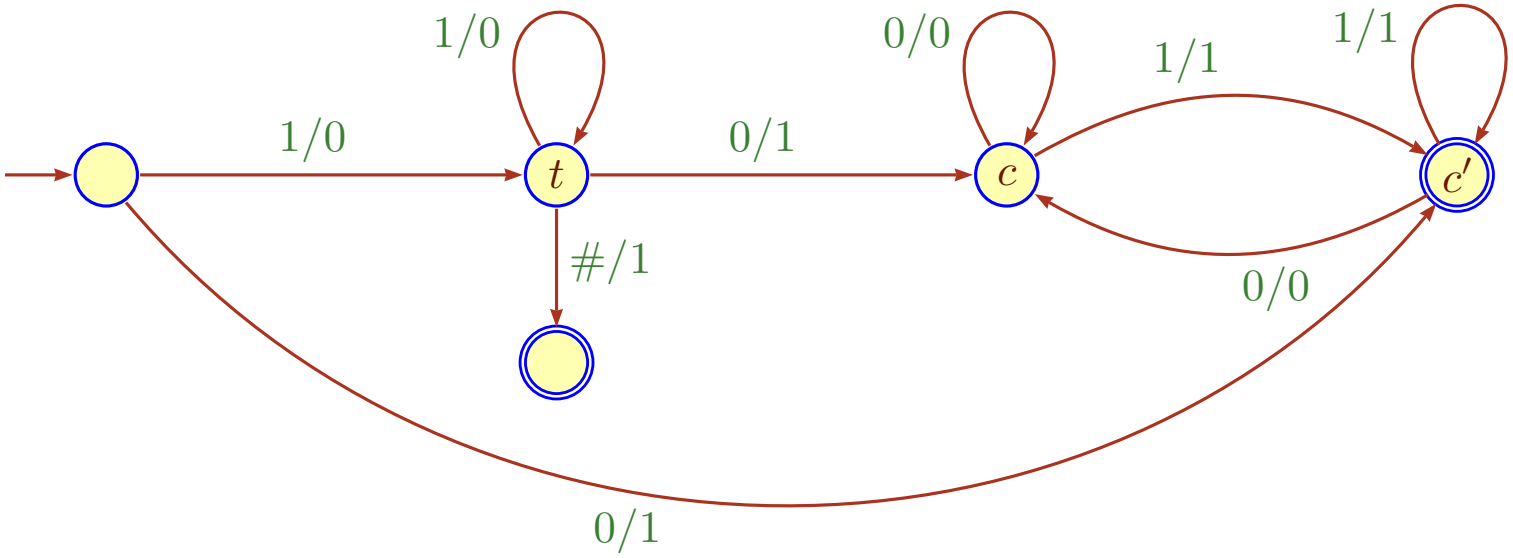
This is our old Mealy machine (insanely called "adding machine" by the group theory people):



Input is supposed to be in reverse base 2.

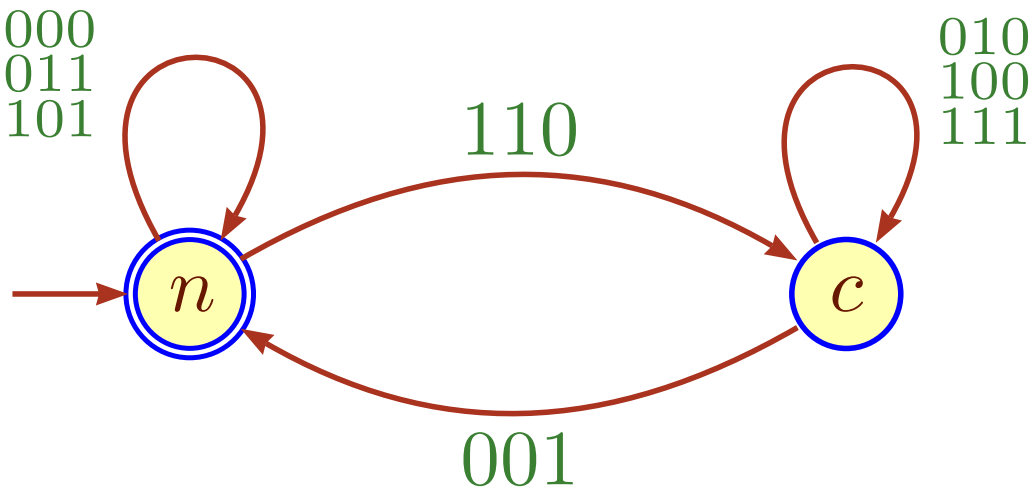
Two problems: this allows for trailing zeros and it implements a cyclic counter: $1^k \mapsto 0^k$.

Compliant Successor



This version is obtained by a bit of surgery on the vanilla machine: handle $1{:}01$, and split state c to eliminate trailing zeros.

Vanilla Addition



This is the logical core of any synchronous transducer for addition, a standard ripple carry adder.

Synchronous Addition

Alas, this transducer ignores our numeration system and requires substantial refinement.

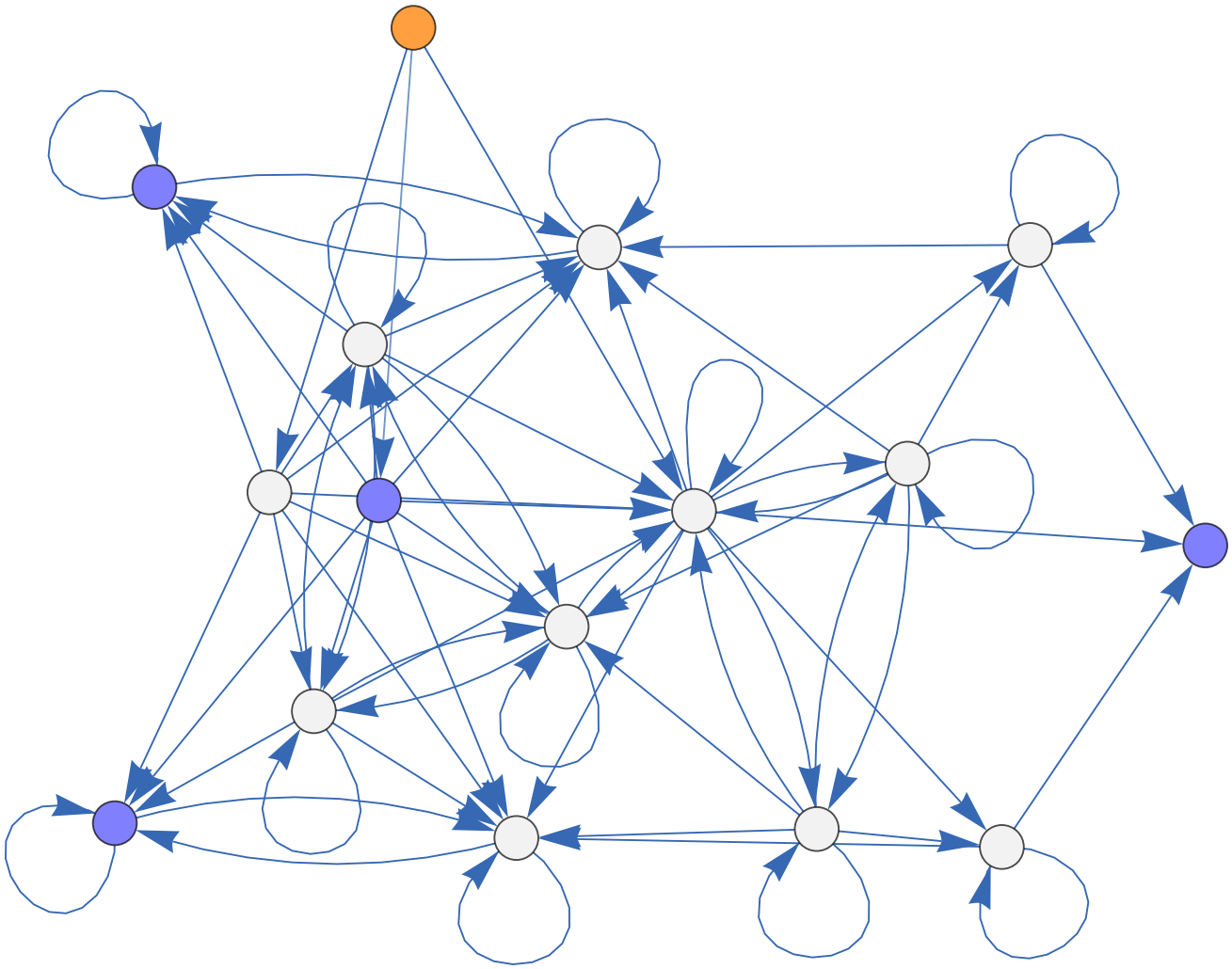
Specifically, we need a synchronous ternary relation $\alpha \subseteq \mathsf{D} \times \mathsf{D} \times \mathsf{D}$ so that

$$\alpha(x, y, z) \iff \mathsf{val}(x) + \mathsf{val}(y) = \mathsf{val}(z)$$

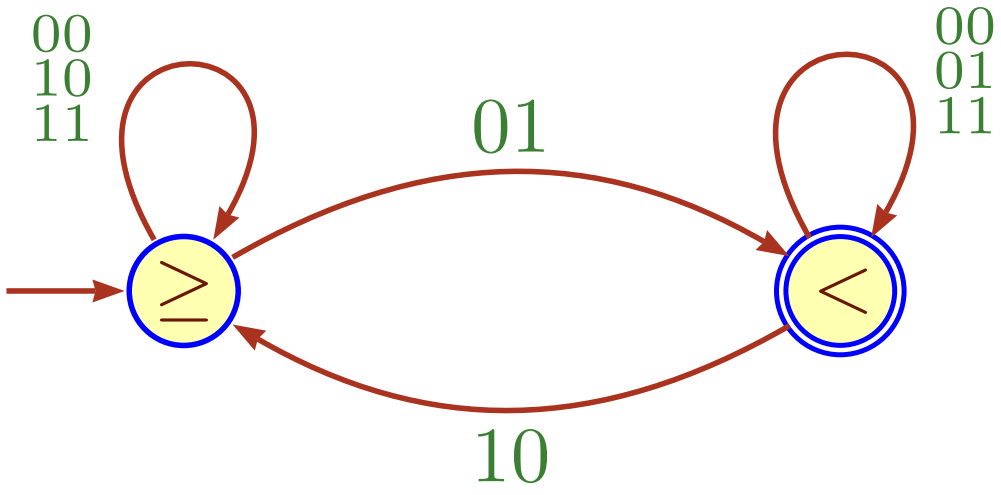
E.g., we need to handle

$$\alpha(0, a, a) \qquad \alpha\big(01\#\#, 111\#, 1001\big)$$

Compliant Addition

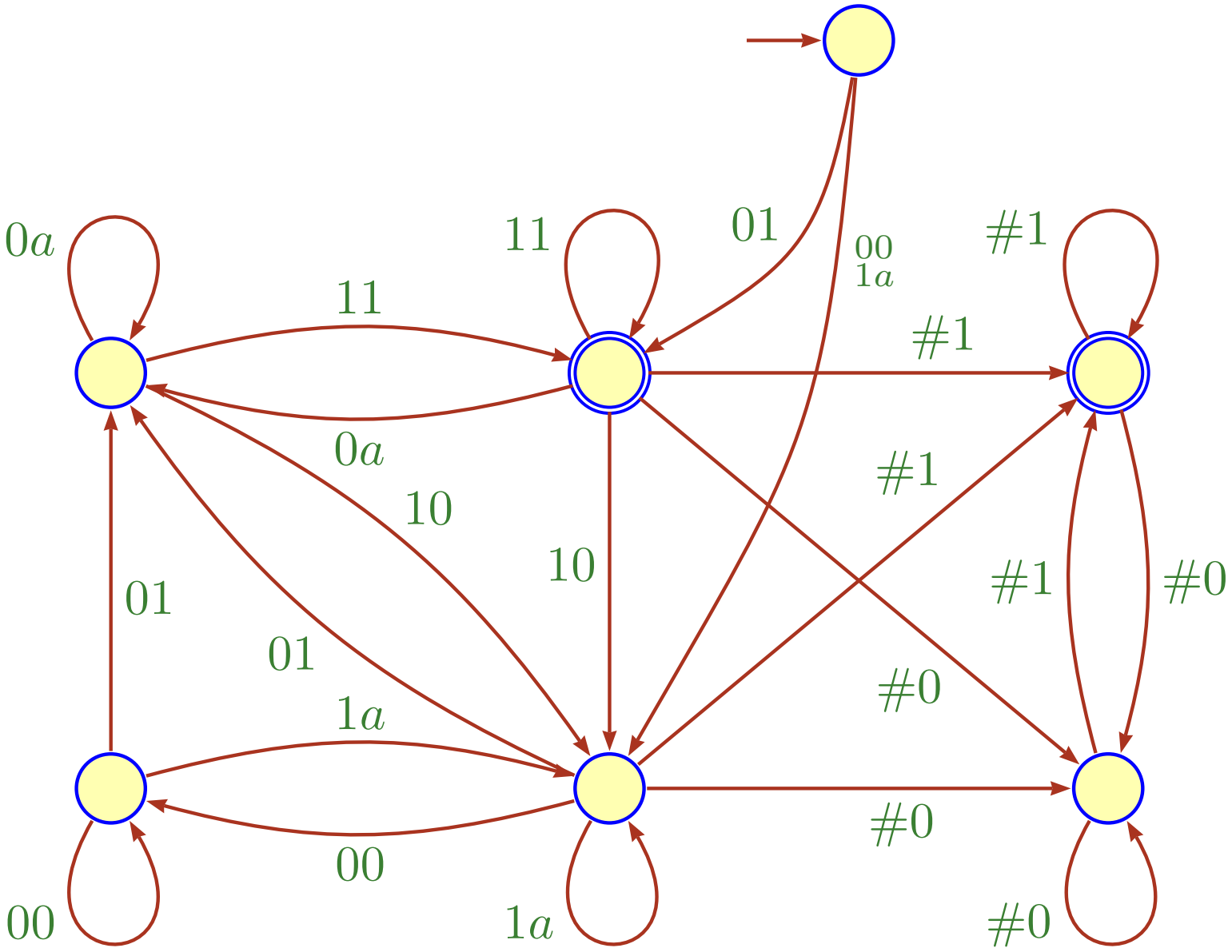


Vanilla Less-Than



The basic comparison transducer: $0{:}1$ leads to acceptance, $1{:}0$ leads to rejection.

Compliant Less-Than



Frobenius Sentence

Here is how the algorithm would handle the sentence

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

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

For the FSM-based proof, let

$$S = \mathsf{span}(3, 5) = 3\mathbb{N} + 5\mathbb{N} = 0, 3, 5, 6, 8, 9, 10, 11, \ldots$$

We need to show that

$$\{\, x \in \mathbb{N} \mid \exists\, y \left(x < y \wedge y \notin S\right) \}$$

is finite.

Cheating

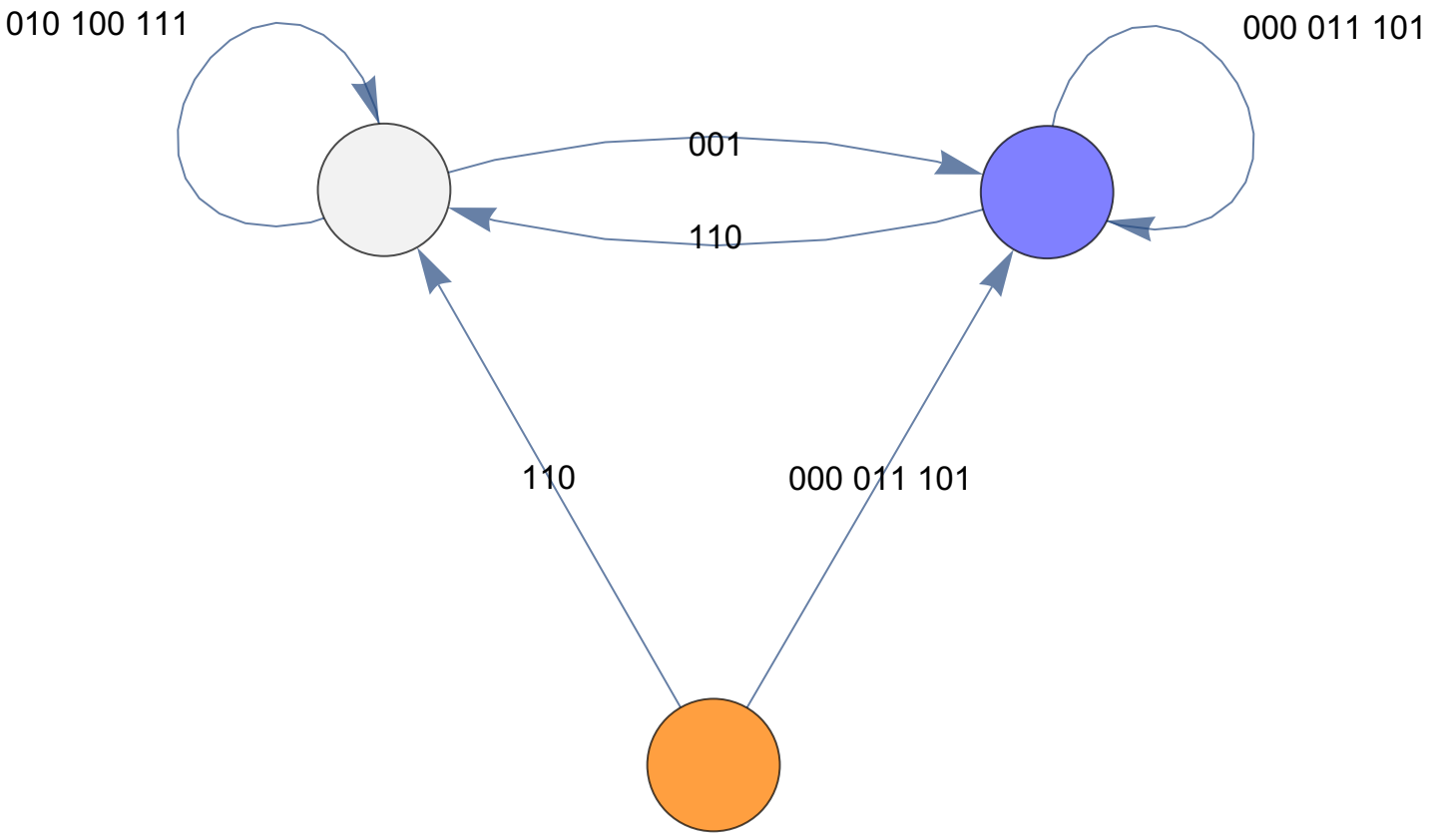
To keep the machines on the next few slides simple, we do not use endmarkers #. Instead we cheat by padding with the right number of 0s.

We do maintain the no-empty-word convention. Still, our machines violate the definition of automaticity.

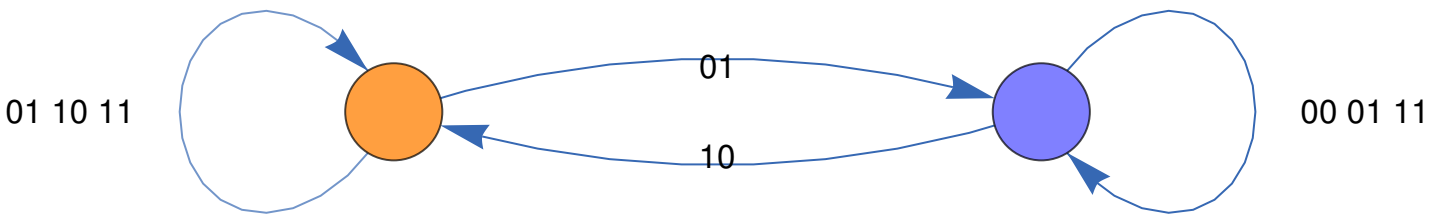
Exercise

Figure out how to fix this and produce compliant machines.

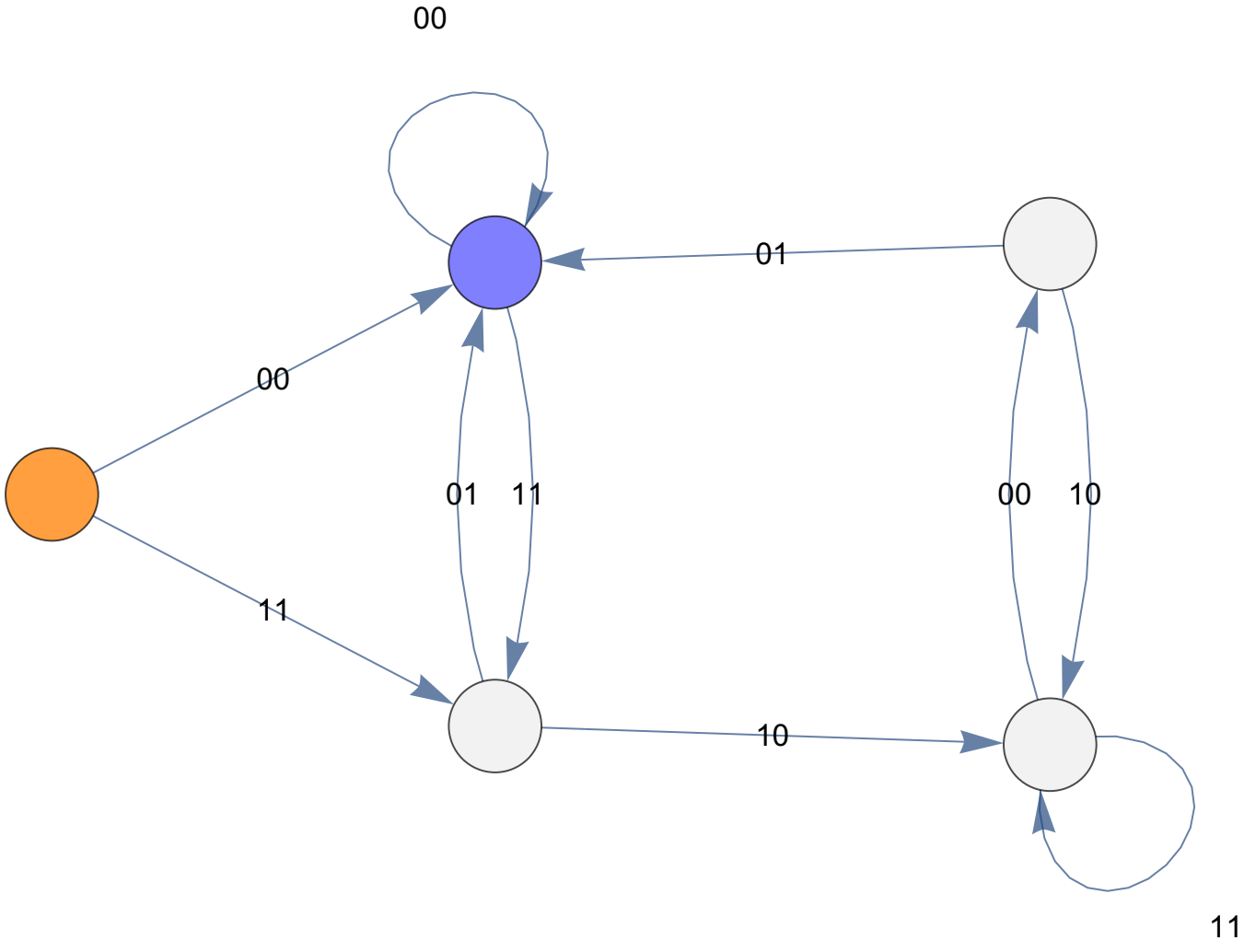
Sloppy Addition



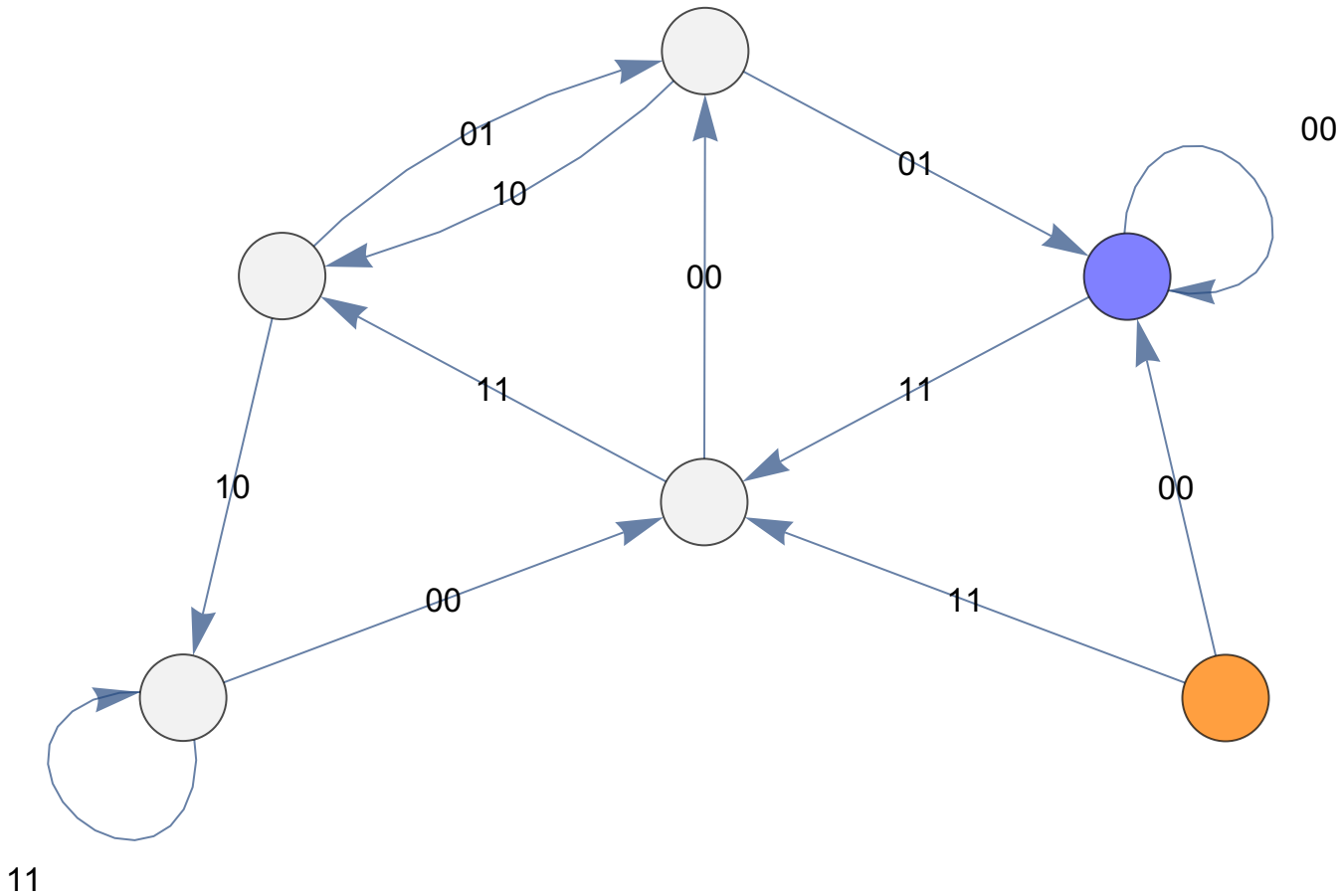
Sloppy Less-Than



Sloppy Multiply by 3



Sloppy Multiply by 5



Span Machine

To handle the span, we use a 5-track machine $\mathcal{S}'$ with the intended variable meaning:

1	u	existential quantifier
2	v	existential quantifier
3	u'	$u' = 3u$
4	v'	$v' = 5v$
5	z	$z = u' + v'$

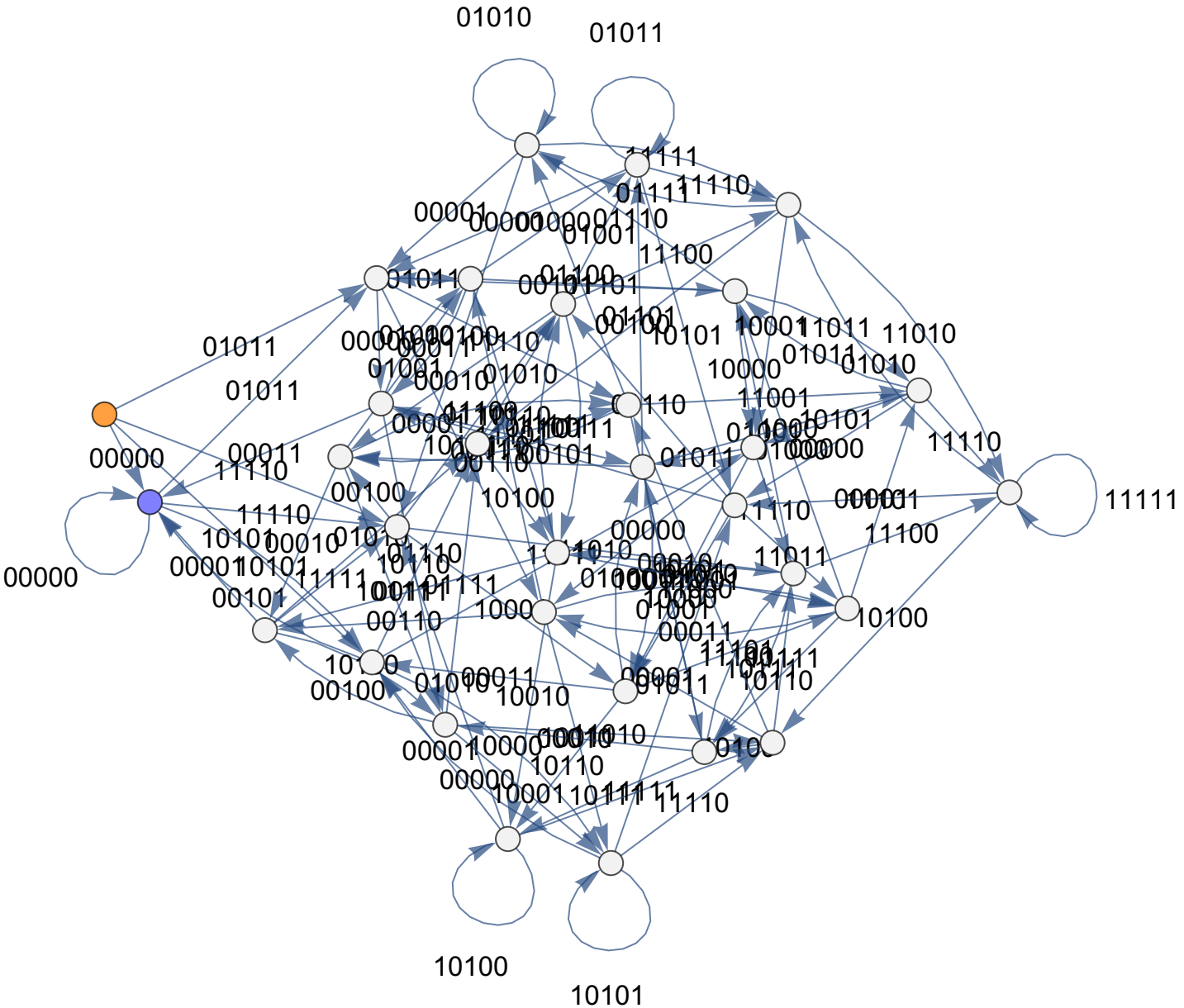
$\mathcal{S}'$ is built from from the multiply-by-3 and a multiply-by-5 machines and and adder.

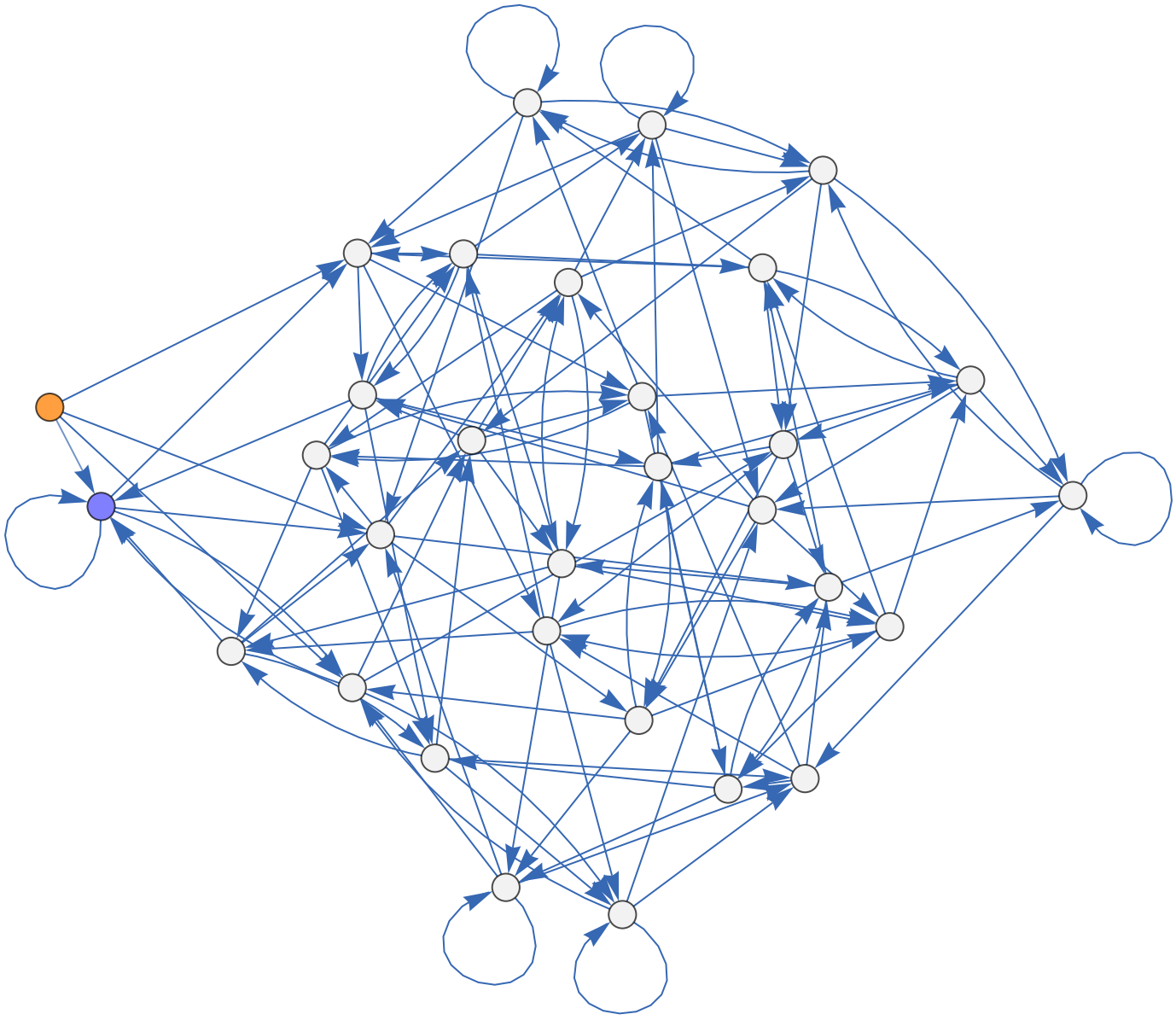
Building Machines

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

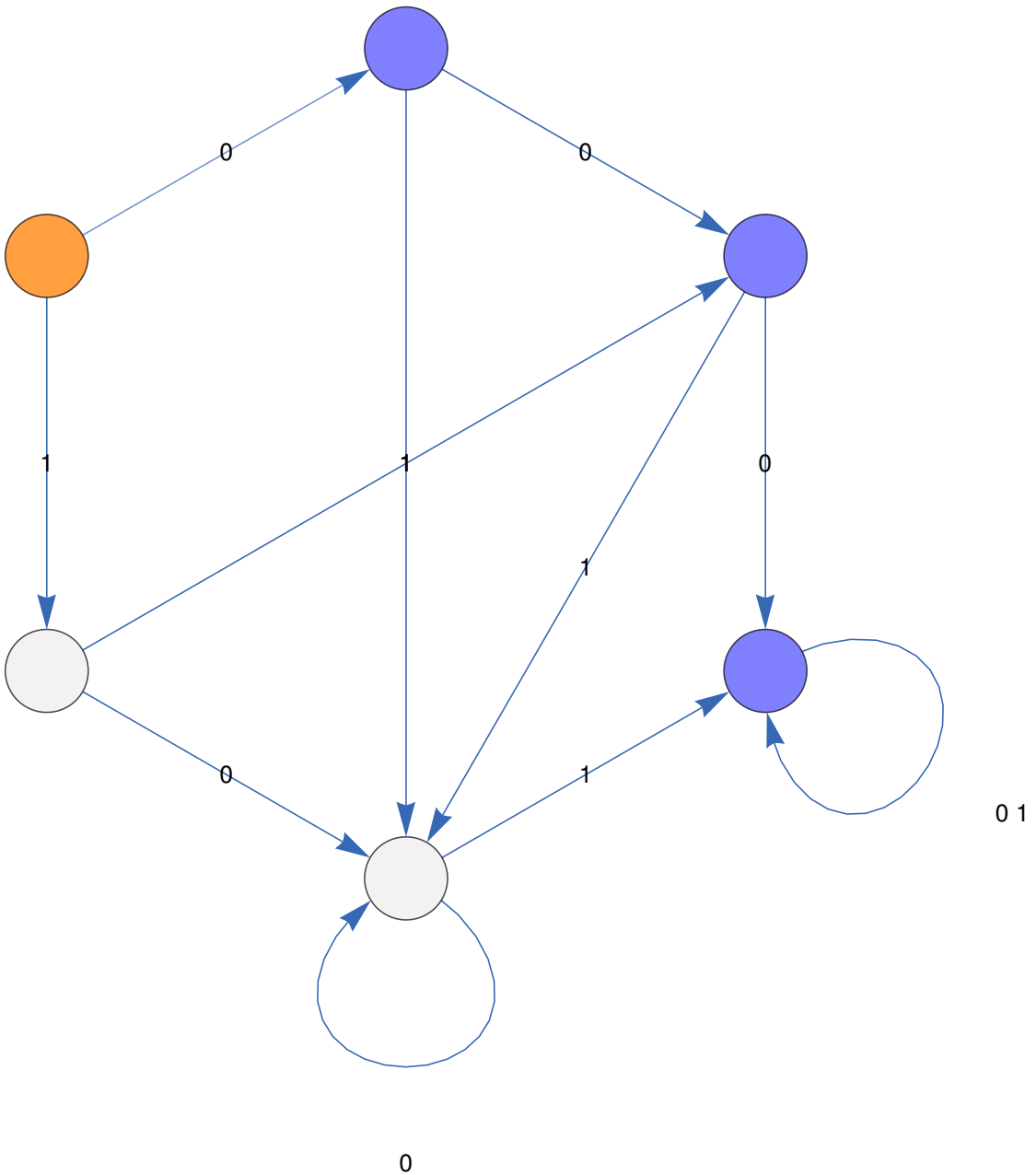
Projecting away all but the z-track produces a machine that recognizes the span of 3 and 5:

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


01010
01011
11111
11110
01111
00001
01001
11100
01011
01100
10101
10001
11011
11010
01011
01010
10000
01011
00010
01010
11001
10101
11110
00011
11110
00000
00100
10100
11111
00000
11100
11011
00000
00101
00110
1000
10100
00011
10010
10110
00001
10110
00000
11110
10100
10101



Minimized $\mathcal{S}$



Onward

It now suffices to check that

$$\Phi \equiv \exists\, x\, \forall\, y\, \big(x < y \Rightarrow y \in S\big)$$

The universal quantifier needs to be rewritten:

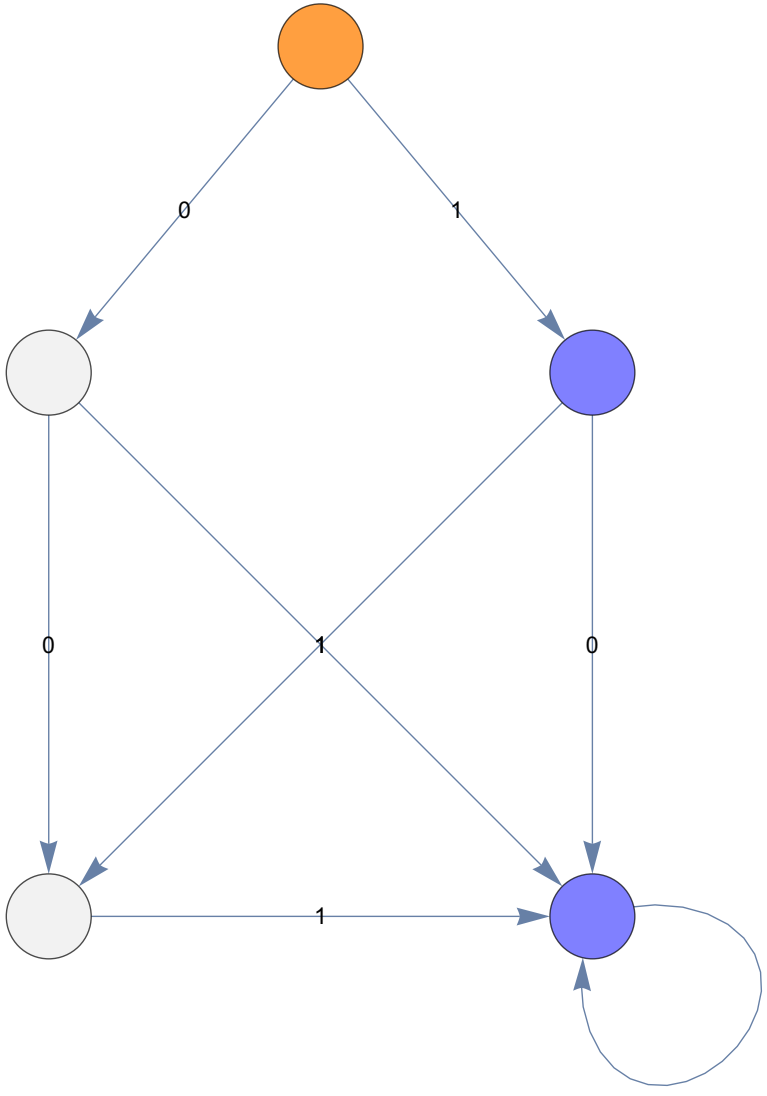
$$\Phi \equiv \exists\, x\, \neg\, \exists\, y\, \big(x < y \wedge y \notin S\big)$$

The $\exists\, y\, \big(x < y \wedge y \notin S\big)$ part is handled by more embeddings:

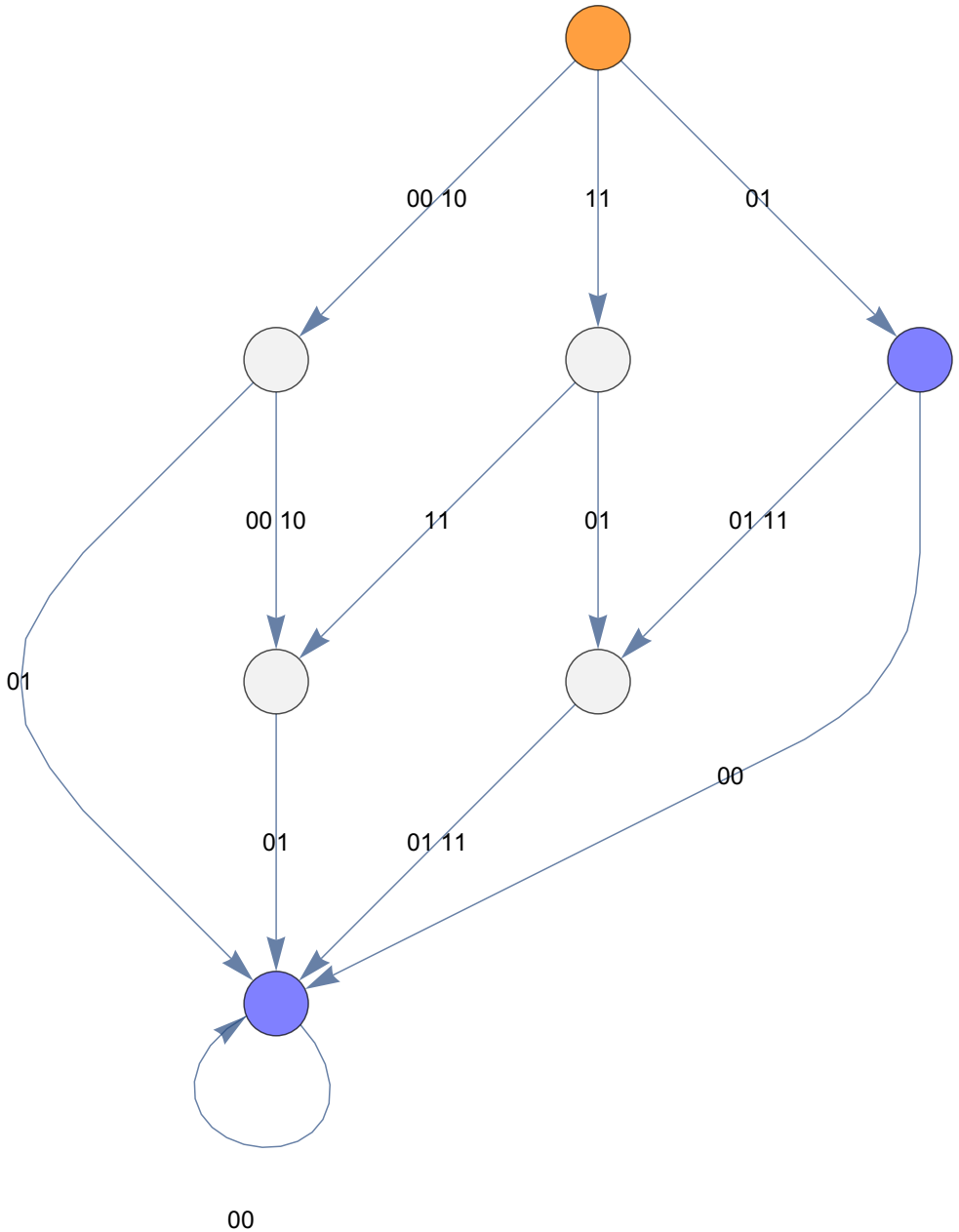
$$\mathcal{C} = \mathsf{emb}^{(2)}_{1,2}\big(\mathcal{A}_<\big) \times \mathsf{emb}^{(2)}_{2}\big(\mathcal{S}_\neg\big)$$

Recall that the complement machine $\mathcal{S}_\neg$ is obtained by intersectiing the full complement machine with D.

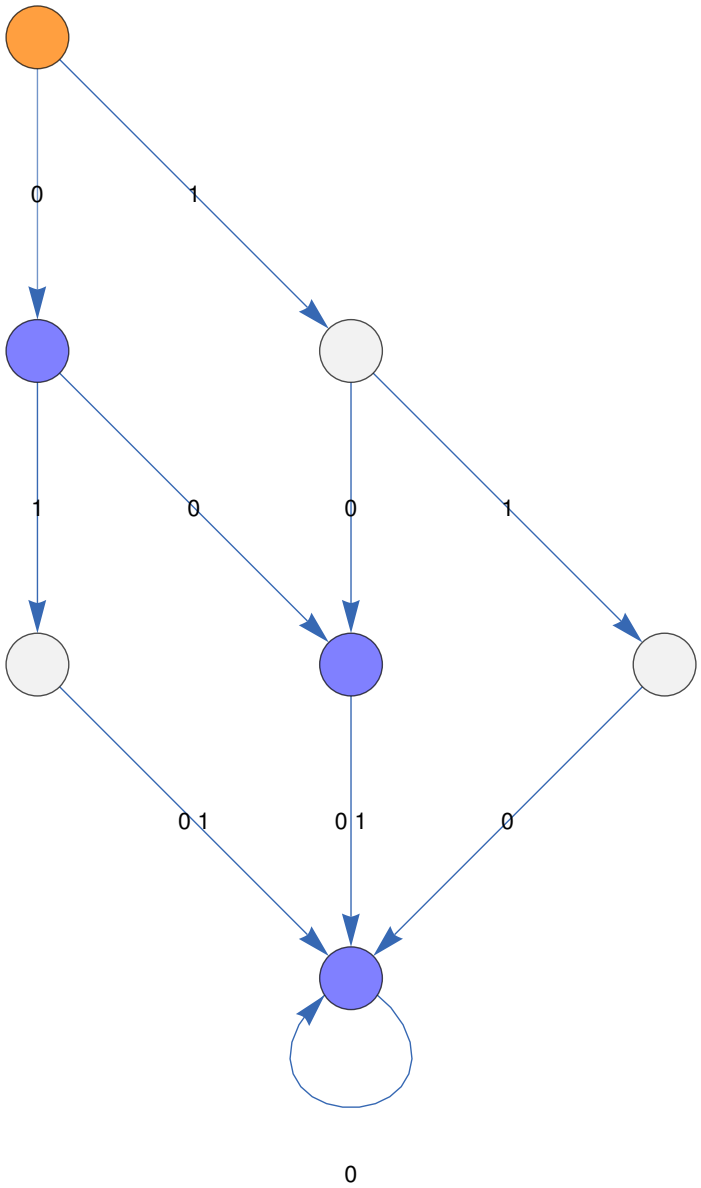
Span Complement



$\mathcal{S}_\neg$ recognizes $1, 2, 4, 7$.


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

Projecting $\mathcal{C}$



least number not accepted: 7

hence witness $x = 7$ works

Isomorphism Problem

Since automatic presentations depend on a names and a value 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 for automatic structures.

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 $\langle \mathbb{N}; \cdot \rangle$ with radix representation
- Free semigroup F_2
- Structures with a pairing function

Warning: Skolem arithmetic is automatic wrto an exotic numeration system based on prime factorizations (in fact, one interprets Skolem arithmetic in Presburger arithmetic).