# Verification and Presburger Arithmetic

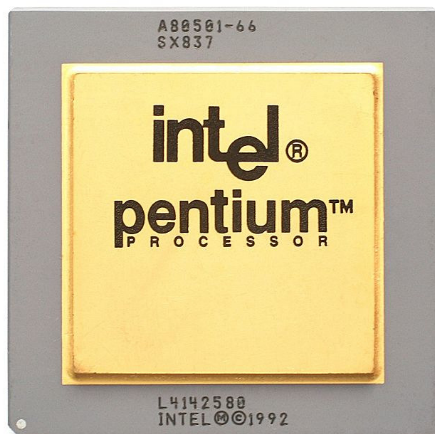
A. Ada & K. Sutner
Carnegie Mellon University

Spring 2018

1 Prelude

- Presburger Arithmetic
- Synchronous Relations
- Deciding Presburger Arithmetic





## Floating Point Error



$$4195835.0/3145727.0 = 1.3338204491362410025 \quad \text{correct}$$
$$4195835.0/3145727.0 = 1.3337390689020375894 \quad \text{pentium}$$

Alternatively

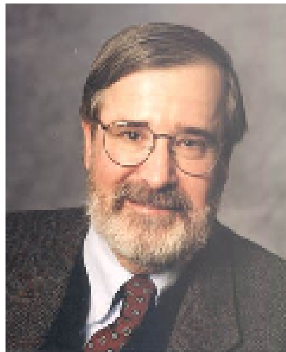
$$4195835.0 - 3145727.0 * (4195835.0/3145727.0) = 0 \quad \text{correct}$$
$$4195835.0 - 3145727.0 * (4195835.0/3145727.0) = 256 \quad \text{pentium}$$

This fatal bug was discovered in October 1994 by number theorist Thomas R. Nicely, doing research in pure math. Intel did not respond well, and lost around $ 500 mill.

Incidentally, this is a software problem at heart.

## Turing Award 2007





If applied, Clarke's model checking methods would have found the bug.

More importantly, they were powerful enough to **prove** that the patch Intel concocted was indeed correct.

## Ariane V

## The Disaster



On June 4, 1996, the maiden flight of the European Ariane 5 rocket crashed about 40 seconds after takeoff. The direct financial loss was about half a billion dollars, all uninsured. An investigation showed that the explosion was the result of a software error.

Particularly annoying is the fact that the problem came from a piece of the software that was not needed during the crash, the Inertial Reference System. Before lift-off certain computations are performed to align the IRS. Normally they should be stopped at $-9$ seconds. However, a hold in the countdown would require a reset of the IRS taking several hours.

## The Cause



To avoid this nuisance, the computation was allowed to continue even after the system had switched to flight mode.

In the Ariane 5, this caused an uncaught exception due to a floating-point error: a conversion from a 64-bit integer (which should have been less than $2^{15}$) to a 16-bit signed integer was erroneously applied to a greater number: the "horizontal bias" of the flight depends much on the size of rocket–and Ariane 5 was larger than its predecessors.

There was no explicit exception handler (since it presumably was not needed), so the entire software crashed and the launcher with it.

## 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

## Software is Applied Logic



There are lots of proposals floating around the problem of constructing correct software. None of them gets around the fundamental problem:

Software is applied logic.

And, logic is hard, even for a super-genius like von Neumann.

**The Good News:** Logic also provides very powerful tools that help in the construction of correct code.


- Prelude

2. Presburger Arithmetic

- Synchronous Relations
- 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

$$A[i,j] = \mathrm{func}(A[i-1,j], A[i,j-1])$$

- 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] = func( 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] = func( 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] = func( 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 $2n - 3$?

The good news is that the index arithmetic involved in the algorithms is quite simple, essentially all we need is addition and order.

That is very fortunate, since arithmetic in general is a tough nut to crack.

## Recall: Matiyasevic



**Theorem (Y. Matiyasevic, 1970)**

*It is undecidable whether a Diophantine equation has a solution in the integers.*

Suppose we wanted to build a reasoning system for ordinary Peano Arithmetic (PA). So we are trying to axiomatize the structure

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

Peano figured out how to do this in 1889; similar arrangements work for $\mathbb{Z}$.

But there is a problem: the terms in (PA) are essentially polynomials with integral coefficients:

$$t(\boldsymbol{x}) = \sum_{\boldsymbol{e}} c_{\boldsymbol{e}}\, \boldsymbol{x}^{\boldsymbol{e}}$$

where $\boldsymbol{x}^{\boldsymbol{e}} = x_1^{e_1} x_2^{e_2} \ldots x_k^{e_k}$.

## Existential Theory



Matiyasevic's theorem implies that it is undecidable whether a formula

$$\exists \boldsymbol{x}\, (t(\boldsymbol{x}) = 0)$$

is true. Never mind more complicated formulae with alternating quantifiers and the like:

$$\forall x\, (\mathsf{even}(x) \land x \geq 4 \Rightarrow \exists u, v\, (x = u + v \land \mathsf{prime}(u) \land \mathsf{prime}(v))$$

If we want to reason about arithmetic, we need to use something weaker than Peano arithmetic.

Here is a radical proposal: let's just forget about multiplication.

## Presburger Arithmetic



We restrict ourselves to the structure

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

The terms in this language are pretty weak:

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

where all the $c$, $c_i$ are constant. This is easy to check by induction.

In particular, we have lost multivariate polynomials–which is a good thing.

## Semilinear Sets



**Definition**

A set $A \subseteq \mathbb{N}$ is linear if $A = \{\, c + \sum c_i x_i \mid x_i \geq 0 \,\}$.

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

Clearly, semilinear sets are definable in Presburger Arithmetic:

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

**Theorem**

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

This is rather 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 $A' \subseteq \{a\}^\star$.

### Theorem

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

### Exercise

*Prove the theorem: figure out what all DFAs over a one-letter alphabet look like.*

## 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.

So 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 $\forall \equiv \neg\exists\neg$.

## The Main Idea



We would like to construct suitable terms $t_i$ that do not contain $z$ such that

$$\exists z \, \varphi(z) \iff \varphi(t_1) \lor \varphi(t_2) \lor \ldots \lor \varphi(t_k)$$

Since an existential quantifier is a kind of disjunction, this is not entirely perplexing. Of course, we need a finite disjunction even when the domain is infinite.

Note that $\Leftarrow$ holds automatically, but $\Rightarrow$ requires work.

This trick is often used in quantifier elimination.

## Boosting the Language



To construct these terms we first have to augment our language a bit (QE fails for the smaller language).

- Add the subtraction operation $x - y$.
- Add a constant $c$ for each integer.
- Add a multiplication function $\lambda x.c * x$ for each integer $c$.
- Add a divisibility predicate $c \mid x$ for each integer $c$.

These additions do not change the expressiveness of our system, they just make it easier to write things down: $3 * x$ instead of $x + x + x$, $3 \mid x$ instead of $\exists\, z\, (z + z + z = x)$.

Note that all terms are still linear despite the additions to the language.

## Using Normal Forms



We may safely assume that $\varphi$ is in disjunctive normal form. Since

$$\exists\, z\, (\varphi_1(z) \lor \varphi_2(z)) \iff \exists\, z\, (\varphi_1(z)) \lor \exists\, z\, (\varphi_2(z))$$

it suffices to remove a quantifier from a conjunction of atomic formulae, say

$$\varphi = \varphi_1 \land \varphi_2 \land \ldots \land \varphi_k$$

where the $\varphi_i$ are quantifier-free.

The $\varphi_i$ will contain $z$ and possibly other free variables (which are bound in the big formula by other quantifiers).

## Cleanup



Cleanup of predicates works like so:

$$\begin{aligned}
t_1 < t_2 \quad &\leadsto \quad 0 < t_2 - t_1 \\
\neg(t_1 < t_2) \quad &\leadsto \quad t_2 < t_1 + 1 \\
t_1 = t_2 \quad &\leadsto \quad t_1 < t_2 + 1 \land t_2 < t_1 + 1 \\
t_1 \neq t_2 \quad &\leadsto \quad t_1 < t_2 \lor t_2 < t_1 \\
\neg(c \mid t) \quad &\leadsto \quad \bigvee_{i=1}^{c-1} c \mid t + i
\end{aligned}$$

Note the rules need to be applied recursively.

## Now What?



We are left with a conjunction that consists of atomic pieces

$$0 < t \qquad c \mid t$$

with linear terms $t$ that may contain the elimination variable $z$.

**Observation:**

$$0 < t \iff 0 < kt \qquad c \mid t \iff kc \mid kt$$

for any positive $k$.

So we can manipulate the coefficients next to $z$ in all terms containing $z$.

## Killing Constants



Let $c_i$ be all the coefficients next to $z$, and let $C$ be their LCM.

Multiply the atomic formula containing coefficient $c_i$ by $C/c_i$. This produces uniform coefficients $\pm C$ everywhere.

So our formula is now equivalent to

$$\exists\, z\, \varphi'(Cz)$$

But that is equivalent to

$$\exists\, z\, (\varphi'(z) \wedge C \mid z)$$

where the coefficients of $z$ in $\varphi'$ are now just $\pm 1$.

Almost there.

## The Vanishing Variable



We can rearrange things a bit to get a formula that looks like

$$\psi(z) = \bigwedge a_i < z \;\wedge\; \bigwedge z < b_j \;\wedge\; \bigwedge d_k \mid z + t_k$$

First, for simplicity, let's ignore the divisibility constraints. Then the last formula is clearly equivalent to $\max a_i + 1 < \min b_j$ which is equivalent to

$$\bigwedge_{ij} a_i + 1 < b_j$$

Et voila: the pesky $z$ is gone.

## Test Terms



This can can also be expressed using test terms:

$$\bigvee_{i=1}^{\alpha} \psi(a_i + 1)$$

And we can handle divisibility constraints

$$\bigvee_{i=1}^{\alpha} \bigvee_{j=1}^{D} \psi(a_i + j)$$

where $D$ is the LCM of all the $d_i$.

The case when there are no lower bounds on $z$ is similar.

## Presburger Arithmetic is Decidable



It follows that first-order logic for arithmetic without multiplication is decidable.

It should be clear from the quantifier elimination process given here that an implementation is somewhat messy.

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

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


- Prelude
- Presburger Arithmetic
3. Synchronous Relations
- Deciding Presburger Arithmetic

## Getting Rid of Proof Theory



The last argument is really an exercise in proof theory: we have shown that a certain theory (augmented Presburger) can establish equivalences

$$\exists z\, \varphi(z) \iff \varphi'$$

This is great, but perhaps there is an alternative approach that does not depend quite so heavily on syntactic details?

An approach grounded in semantics rather than proofs?

## Relational Structures



A relational structure is a first-order structure consisting of a carrier set and a few relations (of arbitrary arities):

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

In other words, we simply do not allow any functions.

This may seem too radical, but we can always fake functions as relations:

$$F(x, y) \iff f(x) = y$$

## White Lie



All true, but note that a purely relational vocubulary makes our formulae a bit more complicated.

For example, consider the formula

$$f(f(x)) = y$$

There really is a hidden quantifier:

$$\exists z\, (f(x) = z \wedge f(z) = y)$$

and so any decision algorithm has to cope with this invisible quantifier.

In a purely relational structure everything is out in the open, we have to write something like

$$\exists z\, (F(x, z) \wedge F(z, y))$$

## FSMs to the Rescue



So the structures we are interested in have the restricted form

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

where

- $A \subseteq \Sigma^\star$ is a regular set of words, and
- $R_i \subseteq A^{k_i}$, a rational relation.

Note that this trivially includes all finite structures.

But we also can easily express infinite sets like $\mathbb{N}$ or $\mathbb{Z}$ in this manner.

## Bummer



The question is whether rational relations

- are expressive enough to write down interesting properties,
- are well-behaved enough for us to decide whether a first-order formula holds over such a structure.

The answer to question 1 is mostly yes.

But question 2 gets a loud NO: we already know that rational relations are not closed under intersection and complement.

## Scaling Back



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

One sledge-hammer restriction is to insist that all the relations are length-preserving. In this case we have, say, $R \subseteq (\Sigma \times \Gamma)^\star$, so we are actually dealing with words over the product alphabet $\Sigma \times \Gamma$.

These can be checked by an ordinary one-tape FSM over this product alphabet:

| $x_1$ | $x_2$ | $\ldots$ | $x_n$ |
|---|---|---|---|
| $y_1$ | $y_2$ | $\ldots$ | $y_n$ |

## 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 often written $x{:}y$.

$x{:}y =$

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

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

## Two Comments 

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

| $a$ | $\#$ | $b$ | $\#$ |
|---|---|---|---|
| $a$ | $b$ | $a$ | $\#$ |

is not allowed.

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

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

Exercise

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

## 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

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

$k$-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. We can always choose $\mathcal{A}$ to be a DFA (though efficiency may be better with nondeterministic machines).

## Again . . .



In a sense, synchronous relations are the most basic examples of relations that are not entirely trivial: given two words $x$ and $y$, we pad out the shorter word to get equal length

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

and then a one-tape DFA reads two symbols $(x_i, y_i)$ or $(\#, y_i)$ at each step.

By contrast, one sometimes refers to arbitrary rational relations as asynchronous: we need two tapes and the two heads can separate arbitrarily far.

## (Counter)Examples



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

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

## A Justification



Our motivation for synchronous relations was taken from length-preserving relations. The reason this works out in the end is the following surprising result.

**Theorem (Elgot, Mezei 1965)**

*Any length-preserving rational relation is already synchronous.*

Note that this sounds utterly fishy: why should we be able to synchronize the two heads to move in lockstep? All we know is that in the end they will have taken the same number of steps.

The proof is quite messy, we'll skip.

## Boolean Operations 

Claim

*Synchronous relations are closed under union, intersection and complement.*

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

$$R = \left(\tfrac{a}{\varepsilon}\right)^\star \left(\tfrac{\varepsilon}{a}\right)^\star$$
$$S = \left(\tfrac{b}{b}\right)^\star$$

Then both $R$ and $S$ are synchronous, but $R \cdot S$ is not (the dot here is concatenation, not composition).

Exercise

*Prove all examples and counterexamples.*

## Synchronous Composition 

On the upside, synchronous relations are closed under composition.

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

Theorem

*If both $R$ and $S$ are synchronous relations, then so is their composition $R \circ S$.*

Exercise

*Prove the theorem.*

## Synchronous Projections



More good news: synchronous relations are closed under projections.

**Lemma**

*Whenever $R$ is synchronous, so is its projection $R'$.*

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.


- Prelude
- Presburger Arithmetic
- Synchronous Relations

4 Deciding Presburger Arithmetic


## The Model Checking Algorithm



So suppose we have implemented all the finite state machines describing Presburger arithmetic (see below).

We are given a sentence $\Phi$ of Presburger arithmetic and want to determine whether it is true.

As always, we may assume that quantifiers use distinct variables and that the given 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.

## Atomic



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

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

Given actual strings for the variables, these can be tested by synchronous transducers $\mathcal{A}_=$, $\mathcal{A}_<$ and $\mathcal{A}_+$ (2, 2, and 3 tracks, respectively).

Strictly speaking, we can also say $x = 0$ and $x = 1$, but that is straightforward to deal with.

## From Atomic to Quantifier-Free



Back to the matrix, the quantifier-free formula

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

with all variables as shown. We construct a $k$-track machine by induction on the subformulae of $\varphi$.

The atomic pieces read from the appropriate tracks and check the corresponding relation using the given transducers.

More precisely, we use variants such as $\mathcal{A}_{=,x,y}$ to check that the strings in the $x$ and $y$ track are equal.

These machines can all be defined over the joint alphabet $\{0, 1, \#\}^k$ (though each machine only needs a few of the bits). Note that the alphabet grows exponentially with $k$, so there is an efficiency problem.

## Boolean Connectives



$\varphi = \psi_1 \wedge \psi_2$ We can build the product of the corresponding machines $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$.

$\varphi = \psi_1 \vee \psi_2$ We take the disjoint union of the corresponding machines $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$ (nondeterminism is inevitable here).

$\varphi = \neg\psi$ Negations may require determinization: if $\mathcal{A}_\psi$ is nondeterministic, we have to convert to a DFA first.

At any rate, we wind up with a composite automaton $\mathcal{A}_\varphi$ for the whole matrix:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{\, u_1{:}u_2{:}\ldots{:}u_k \in (\mathbf{2}^k_\#)^\star \mid \mathcal{C} \models \varphi(u_1, u_2, \ldots, u_k) \,\}$$

## Quantifiers



It remains to deal with quantifiers. We will work on the innermost quantifier, say

$$\exists\, z\, \varphi(z)$$

We have a machine $\mathcal{A}_\varphi$ that has a track for variable $z$.

> Simply erase the $z$-track from all the transition labels.

This corresponds exactly to existential quantification, done!

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

This is all permissible, since projections and negations do not disturb automaticity.

## Finale Furioso



In the process of removing quantifiers, we lose one track at each step (but remember that universal quantifiers cause quite a bit additional activity).

In the end, we wind up with an unlabeled transition system: just a directed graph, all edge-labels are gone. This transition system has a path from $I$ to $F$ iff the original sentence $\Phi$ is valid.

So the final test is nearly trivial: plain DFS works fine.

Alas, it does take a bit of work to construct this digraph in the first place.

## 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. 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.

## Implementing $\mathfrak{N}_-$ 

There are several choices for the representation of natural numbers:

- binary (MSD first)
- binary (MSD first), no leading $0$'s
- reverse binary (LSD first)
- reverse binary (LSD first), no trailing $0$'s
- $\varepsilon$ denotes/does not denote $0$

They all work equally well, but note that the machines implementing the relations will be slightly different.

Let's assume options 4 and no $\varepsilon$.

## Relations 

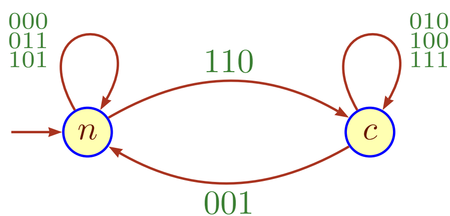
- addition as a ternary relation $\alpha$
- "constants" zero and one as unary relations
- order less as a binary relation

These are all synchronous, regardless of how we represent the naturals.

Exercise

*Check that this is really true.*

## Addition 

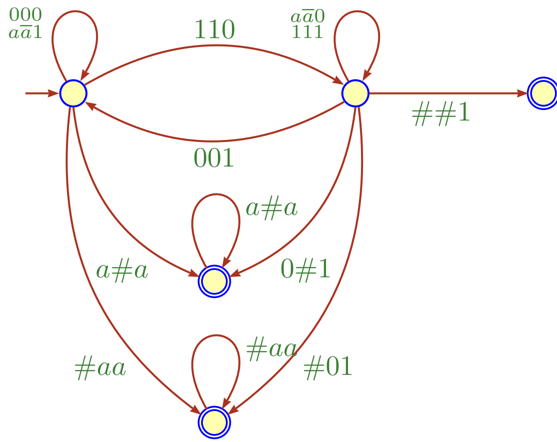



This is the core of the transducer for addition, with states "no carry" and "carry."

But note that this is not the automaton we need.

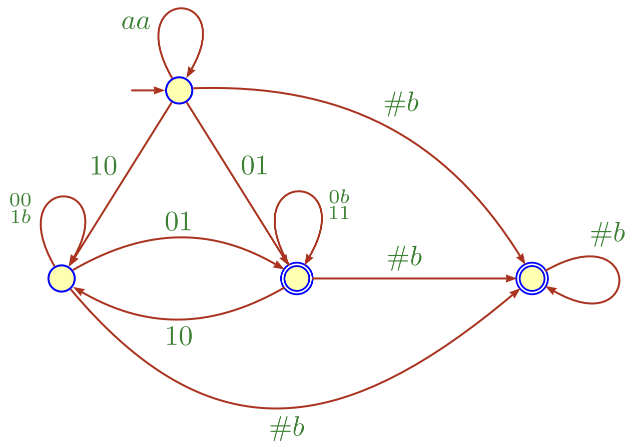
## Cum Endmarker






## Less-Than






## A Sentence



Consider the sentence

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

Thanks to our brilliant choice of coefficients, $\Phi$ is actually true.

With a view towards processing, let's rewrite $\Phi$ in prenex normal form and get rid of the implication:

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

## The Automaton



We are going to construct a 4-track automaton $\mathcal{A}$, one track for each of the variables $x$, $y$, $u$ and $v$ in $\Phi$.

First we need to build an automaton for the matrix $y < x \lor 3 * u + 5 * v = y$.

**Warning:** We are living in a relational world, we have to rephrase the second part as

$$\exists\, z_1, z_2, z_3 \left(\mathsf{mult}_3(u, z_1) \land \mathsf{mult}_5(v, z_2) \land \mathsf{add}(z_1, z_2, z_3) \land \mathsf{eq}(z_3, y)\right)$$

So we first build a 7-track machine $\mathcal{A}'_e$ using products of the canonical machines for multiplication by 3 and 5, for addition and for equality.

Then we project away the tracks $z_1$, $z_2$, $z_3$ and wind up with a 4-track machine $\mathcal{A}_e$.

## Onward



We take the disjoint union of $\mathcal{A}_e$ and $\mathcal{A}_<$ to get a 4-track machine $\mathcal{A}_\varphi$ that checks the matrix of $\Phi$.

We eliminate the $u$ and $v$ tracks to account for the innermost existential quantifiers $\exists\, u, v$ .

Then we rewrite the universal quantifier as $\neg\exists\, y\, \neg$ and eliminate as usual.

Right now our automaton $\mathcal{A}$ accepts all $x$ that are witnesses for $\Phi$.

The last step is to eliminate $\exists\, x$ and obtain the answer Yes. Note that this is essentially an Emptiness test for finite state machines.