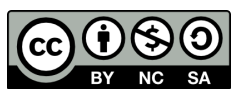
CDM

Rewrite Systems

Klaus Sutner

Carnegie Mellon University

Spring 2025



In the case of languages, the usual dichotomy between analysis and synthesis takes the following form:

Recognition Develop an algorithm that, given a word, determines whether it belongs to the language.

Generation Develop a method that allows one to generate all words in a language in some systematic fashion.

From a computability perspective, recognition requires decidability, but it is far from clear what a solution to the generation problem would look like. For example, if L is infinite the enumeration has to run forever, so it is different from a standard algorithm.

This becomes clear when we try to dress up these questions as a standard combinatorial problem:

Problem:	**Recognition** L
Instance:	A language $L \subseteq \Sigma^\star$, a string $x \in \Sigma^\star$.
Question:	Is $x \in L$?

Fine, but the analogous problem for generation looks a bit weird:

Problem:	**Generate** L
Instance:	A language $L \subseteq \Sigma^\star$.
Solution:	Enumerate all $x \in L$.

The good news is that we have already encountered a very similar situation in classical computability theory: decidability versus semidecidability, and semidecidability is equivalent to being recursively enumerable. So, in general, generation will be more powerful than recognition.

However, if the languages in quesion are fairly simple we won't have some arbitrary Turing machine grinding away. So there is a good chance that we may be able to connect recognition and generation very tightly. Specifically, we will use rewrite systems to generate languages.

But the main idea really goes back to the 1930s, and even earlier.

Axel Thue (1863–1922)

Thanks to CS, rewrite systems are now an active research area, a century after their invention.

Just to be clear, we are facing two issues:

- Given a decision algorithm, extract an enumeration algorithm.
- Given an enumeration algorithm, convert it into a decision algorithm.

Problem one is easy in principle: enumerate all objects, and use the decision algorithm to select the appropriate ones. Of course, efficiency is another matter entirely.

The second problem is highly important in the RealWorld^{TM}: the enumeration algorithm is the specification of a programming language, and we want to convert it into a parser.

Definition

A rewrite system (r/w-system) is a pair

$$\mathcal{R} = \langle \Sigma, \mathcal{P} \rangle$$

where Σ is an alphabet and $\mathcal{P} \subseteq \Sigma^\star \times \Sigma^\star$ is a finite set of productions, also called r/w rules.

We often write productions with a short arrow, as in

$$u \to v$$

Rewrite systems are also called semi-Thue systems or reduction systems.

If Σ is clear from context, we simply write $\mathcal{P}$ for the whole system.

The main idea is very simple: a string $U = xuy$ can be rewritten to $V = xvy$ whenever $u \to v$ is in $\mathcal{P}$. The LHS u is called a handle. Note: there may be multiple handles in a string; worse, they may overlap.

Technically, we define a derivation relation on $\Sigma^\star$. First, a one-step derivation:

$$xuy \to_{\mathcal{P}} xvy \iff u \to v \text{ in } \mathcal{P}$$

Full derivation $\xrightarrow{*}_{\mathcal{P}}$ is then the reflexive transitive closure of one-step. We drop the subscripts if the system in question is clear.

Claim

For any rewrite system $\mathcal{P}$, the derivability relation $\xrightarrow{*}$ is semidecidable.

Given a r/w-system $\langle \Sigma, \mathcal{P} \rangle$ fix some word $w \in \Sigma^\star$, sometimes called the axiom.

Then we obtain a language by collecting all strings derivable from w:

$$\mathcal{L}(\mathcal{P}, w) = \{\, x \in \Sigma^\star \mid w \xrightarrow{*} x \,\}$$

By definition, $\mathcal{L}(\mathcal{P}, w)$ is recursively enumerable: it is not difficult to systematically generate all derivations in the system.

Exercise

Describe a method to generate all derivations in a r/w-system.

Rewrite rules are naturally directed: u can be replaced by v, but not necessarily the other way round. If the rules come from equations, it is natural (though algorithmically problematic) to include both directions: Sometimes (in particular in algebra) one wants to think of the rules as equations: $x \approx y$ turns into $x \to y$ and $y \to x$. In this case one speaks of a Thue system.

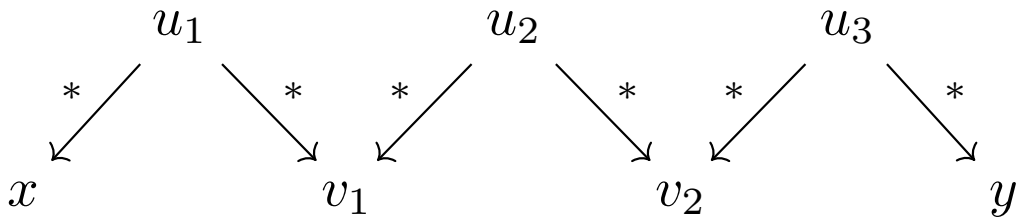
Correspondingly, define the symmetric closure of $\mathcal{P}$ to be

$$\mathcal{P}^s = \mathcal{P} \cup \mathcal{P}^{\mathsf{op}} = \{\, u \to v \mid u \to v \in \mathcal{P} \vee v \to u \in \mathcal{P} \,\}$$

Definition

The Thue congruence generated by $\mathcal{P}$ is the finest equivalence relation containing $\mathcal{P}$. In sybmols: $x \overset{*}{\longleftrightarrow}_{\mathcal{P}} y$.

Two strings are congruent modulo $\mathcal{P}$ iff $x \overset{*}{\longleftrightarrow}_{\mathcal{P}} y$.



The generic situation for two congruent words $x \overset{*}{\longleftrightarrow} y$: they are connected by a zig-zag, an alternating path of back-and-forth reductions.

If you prefer, you can think of $\mathcal{P}$ as a compact description of an infinite graphs over the vertex set $\Sigma^\star$: the edges are given by the one-step relation $x \to_{\mathcal{P}} y$.

The transition from $\mathcal{P}$ to $\mathcal{P}^s$ is then similar to turning a digraph into a ugraph by ignoring the direction of the edges.

In particular for symmetric Thue systems, the equivalence relation $x \overset{*}{\longleftrightarrow} y$ corresponds to the connected components of the corresponding ugraph.

In a standard rewrite system, the replacements can be made anywhere in the string: a factor u is replaced by v. This is arguably the most natural way to define rewrites.

But, there is an alternative that is particularly attractive in connection with efficient algorithms: we can allow rewrites to take place only at the beginning of the string. These systems are referred to as prefix rewrite systems.

In particular when the handles are unique, one can simply read the string from left to right, and replace a prefix whenever the opportunity arises.

More later.

Example: Erasing Things

Consider alphabet $\{a, b\}$ and $\mathcal{P}$: $ab \to \varepsilon$, $ba \to \varepsilon$. Let $f(x) = |x|_a - |x|_b$.

The claim is that $\mathcal{P}$ recognizes balanced words in the following sense:

Claim

$x \xrightarrow{*} \varepsilon$ *iff* $f(x) = 0$.

Proof.

Every application of $\to_{\mathcal{P}}$ preserves f, so the implication left to right is obvious.

For the other direction suppose $x \neq \varepsilon$. Since $f(x) = 0$ we must have $x = uabv$ or $x = ubav$. Either way, $x \to_{\mathcal{P}} uv$, done by induction.

□

$$\mathcal{P}: \quad abc \to ab, bbc \to cb$$

Here are the first few steps applying the Thue system $\mathcal{P}^s$ to abb. We follow all possible rewrites.

abb

$abcb$

$abb, abbbc, abccb$

$abcb, abcbbc, abcccb$

$abb, abbbc, abccb, abbbcbc, abccbbc, abcccb$

$abcb, abcbbc, abcccb, abbbbbcc, abcbbcbc, abcccbbc, abccccb$

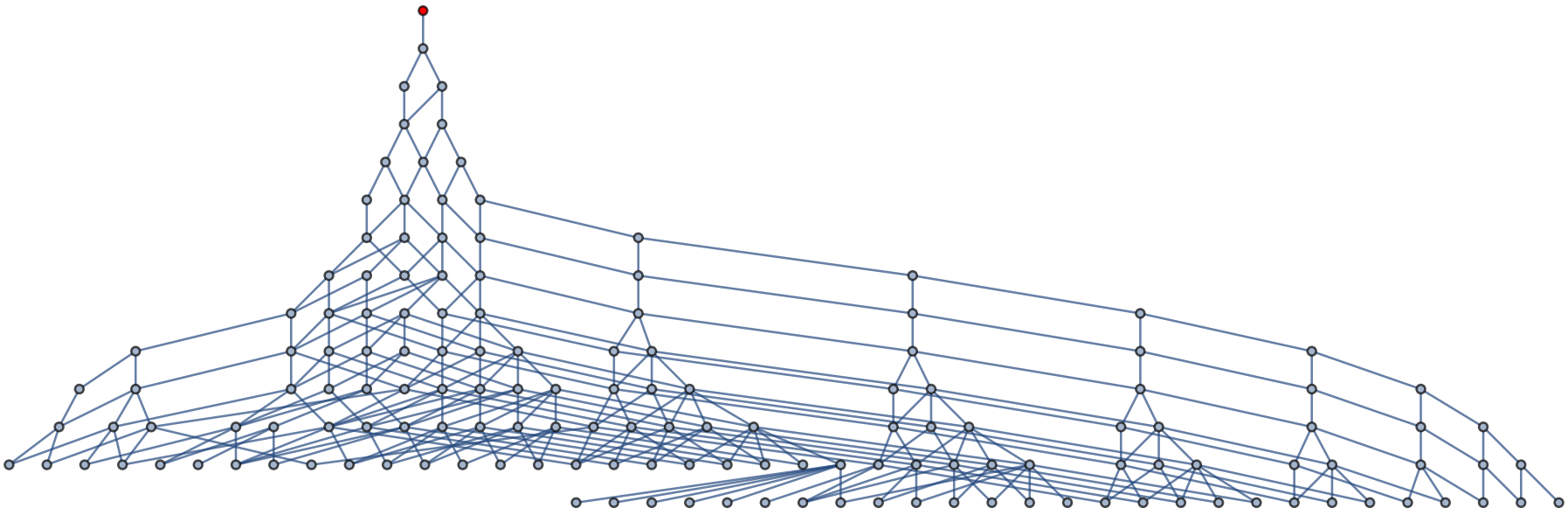
$abb, abbbc, abccb, abbbcbc, abccbbc, abcccb, abbbcbcbc,$
$abcbbbbcc, abccbbcbc, abcccbbc, abccccccb$

This seems fairly chaotic, but if one focuses on strings of the form $a^\star b^\star c^\star$ there is a pattern. Write L for all the strings derivable from abb. Then

$$L \cap a^\star b^\star c^\star = \{\, ab^{2^n+1}c^n \mid n \geq 0 \,\}$$

In a sense, this r/w-system computes the function $n \mapsto 2^n + 1$.

Thus, very simple rewrite systems can easily produce fairly complicated languages.



The first 150 words generated by the system, abb is the one on top.

Derivability is semidecidable, and in general, that is all we can say: it may well be undecidable.

To see why, consider configurations of a Turing machine $\mathcal{M}$ be written as usual as strings of the form

$$C = b_m b_{m-1} \dots b_1 \, p \, a_1 a_2 \dots a_n \in \Gamma^\star \, Q \, \Gamma^\star$$

Clearly, the one-step relation of $\mathcal{M}$ can be described as a rewrite system $\mathcal{P}$, so $\stackrel{*}{\longrightarrow}_{\mathcal{P}}$ corresponds to computations of the TM.

Thus, a decison algorithm for $\stackrel{*}{\longrightarrow}_{\mathcal{P}} y$ would solve the Halting problem for $\mathcal{M}$.

But, for less general rewrite systems there are lots of interesting results.

Here is how a CAS uses rewriting to simplify a mathematical expression:

```
D[ 2 Cos[3 x - Pi], x ] - 6 Sin[3 x]   ==>  0
```

A (partial) trace of the corresponding evaluation:

$$
\begin{aligned}
(((3x-\pi,-\pi+3x),\cos(-\pi+3x),-\cos(3x)),&\\
2(-\cos(3x)),2(-1)\cos(3x),-2\cos(3x),-2\cos(3x)),&\\
\partial_x(-2\cos(3x)),6\sin(3x),-(6\sin(3x)),-6\sin(3x),&\\
-6\sin(3x),6\sin(3x)-6\sin(3x),-6\sin(3x)+6\sin(3x),&\\
0&
\end{aligned}
$$

There are even whole programming languages based entirely on the notion of rewriting, see for example

http://code.google.com/p/pure-lang/

> *Pure is a functional programming language based on term rewriting. This means that all your programs are essentially just collections of symbolic equations which the interpreter uses to reduce expressions to their simplest ("normal") form.*

```
let X = [x,x^2,x^3];
reduce X with x = u+v end; // yields [u+v,(u+v)^2,(u+v)^3]
```

We have a collection $\mathcal{T}$ of terms and a finite list of rewrite rules on these terms. We use the rules to simplify the terms: we can rewrite an arbitrary term t into a normal form $\nu(t)$.

Then, in order to check whether two terms s and t denote the same object, we rewrite both s and t into normal form and check that $\nu(s) = \nu(t)$.

The last identity is plain equality of terms and is trivial to check.

In practice, one needs the rewrite rules to have a few special properties:

- The precise order in which the rules are applied should not matter.
- Whatever rules we apply, after finitely many steps the rewrite process should terminate (we arrive at a term without descendants).

Without these sanity conditions, things get tricky. Lack of termination is obviously a disaster, but multiple, irreconcible reductions are also ruinous.

We need to formalize this approach and develop algorithms.

Definition

z is a descendant of x iff $x \xrightarrow{+}_{\mathcal{P}} z$.
$x \in \Sigma^\star$ is irreducible iff it has no descendants.
We write $\mathrm{Irr}_{\mathcal{P}}$ for the set of all irreducible terms.

A normal form for x is an irreducible word x' such that $x \xrightarrow{*}_{\mathcal{P}} x'$.
If the normal form is unique, we often write $\nu(x)$.

Normal forms are mostly useful if they are unique: we can use $\nu(x)$ as the *name* or *representative* of x modulo $\mathcal{P}$.

Lemma

Let $\langle \Sigma, \mathcal{P} \rangle$ be a finite rewrite system. Then the set of irreducible words is regular.

Proof.

Irreducibility only depends on the left hand sides of productions $u \to v$, so let u_1, u_2, ..., u_n be a list of all LHS. Then a word x is irreducible iff it is not in $\Sigma^\star \{u_1, \ldots, u_n\}^\star \Sigma^\star$. □

Any generic regex-to-fsm algorithm can construct a nondeterministic fsm for the "forbidden factors" language. Or we could use the Aho-Corasick dictionary algorithm to check for forbidden factors.

In fact, we can do slightly better than that with a little preprocessing.

Proposition

A DFA recognizing Irr *can be constructed in polynomial time.*

Proof.

We may safely assume that ε is not a LHS and that no LHS $u_1, \ldots, u_n$ is a factor of another. Define a partial DFA as follows. States Q are all proper prefixes of the u_i, ε being the initial state; all states are final. The transition function is given by

$$\delta(x,a) = \begin{cases} z & xa \text{ irreducible, } z \text{ longest suffix of } xa \text{ in } Q, \\ \uparrow & xa \text{ reducible.} \end{cases}$$

Correctness by a straightforward induction. □

Another Example

Again let $\Sigma = \{a, b\}$ and

$$\mathcal{P}\text{:} \quad aaaa \to \varepsilon, bb \to \varepsilon, ba \to aaab$$

Burning Question: Is there a normal form? Is it unique?

By the first two rules, any x can be written as

$$a^{\alpha_1} b^{\beta_1} a^{\alpha_2} b^{\beta_2} \ldots a^{\alpha_n} b^{\beta_n}$$

where $0 \leq \alpha_i < 4$ and $0 \leq \beta_i < 2$ and only α_1 and β_n may be 0.

This suggests a normal form $a^{\alpha} b^{\beta}$.

If $n = 1$ we're done.

Otherwise apply the last rule to get

$$a^{\alpha_1+3\alpha_2} b^{\beta_1+\beta_2} \ldots a^{\alpha_n} b^{\beta_n}$$

Reductions by the first two rules produces an expression as above, except that n must have decreased at least by 1.

Done by induction.

Dihedral Group

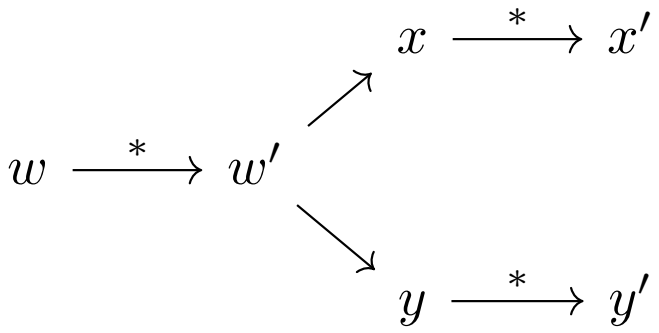
We are really dealing with the dihedral group D_4: rigid motions of a square in the plane.

In this interpretation, a corresponds to a rotation by $\pi/2$ and b corresponds to a reflection (horizontal, vertical or diagonal, it does not matter).

As a geometric operation, $ba = aaab = a^{-1}b$.

Check it out.

The rewrite relation is usually nondeterministic, so an attempt at reducing a term to normal form may fork into to two separate chains.



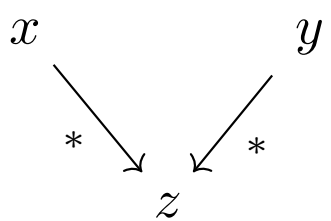
This is particularly tricky when, at the fork, the two chosen handles overlap, so order of application matters greatly: there is no easy fix by duplicating the replacement taken in the other chain.

Ideally, we would like to be able to apply the rewrite rules in any order whatsoever.

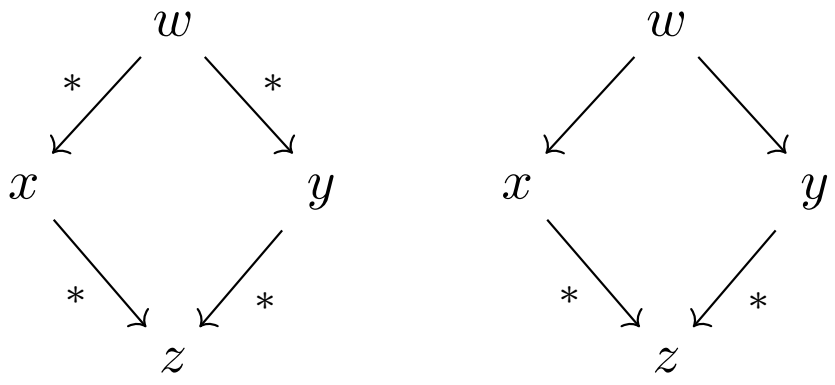
Definition

Two strings x and y are confluent iff there exists a z such that $x \xrightarrow{*} z$ land $y \xrightarrow{*} z$.
The rewrite system is (locally) confluent iff any two words with a (direct) common ancestor are confluent.

We will write $x \downarrow y$ to indicate that x and y are confluent:



So in a confluent system we have for any $w, x, y \in \Sigma^\star$ such that $w \xrightarrow{*} x$ and $w \xrightarrow{*} y$, there is a $z \in \Sigma^\star$ such that $x \xrightarrow{*} z$ and $y \xrightarrow{*} z$.



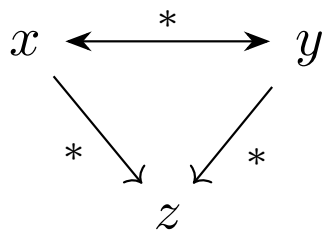
confluent locally confluent

In a confluent r/w-system, nondeterministic choices are not a major problem (though they may influence the length of a reduction).

Definition

A r/w-system is Church-Rosser iff

$$x \overset{*}{\longleftrightarrow} y \quad \text{implies} \quad x, y \text{ are confluent}$$



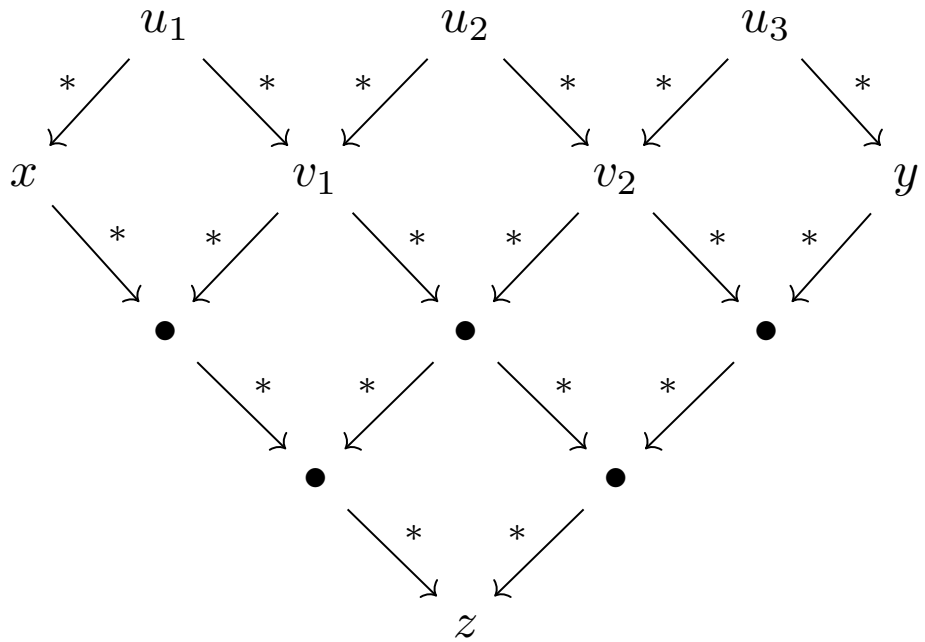
In other words, two terms that are congruent modulo $\mathcal{P}$ already are confluent.
In cryptic symbols:

$$x \overset{*}{\longleftrightarrow} y \Rightarrow x \downarrow y$$

Theorem

A r/w-system is Church-Rosser iff it is confluent.

Proof. LtR is obvious. RtL by picture:



□

Confluence is not enough to guarantee the existence of a unique normal form: the chain of reductions might fail to terminate.

Definition

A rewrite system is terminating or Noetherian if there are no infinite rewrite chains

$$x_0 \to x_1 \to x_2 \to \ldots \to x_n \to x_{n+1} \to \ldots$$

Note that in an algebraic context we cannot simply symmetrize an identity $s \approx t$ to two rules $s \to t$ and $t \to s$ if we want termination.

More generally, any cycle $x \xrightarrow{+} x$ is fatal, the derivation graph of $\mathcal{P}$ must be acyclic.

Theorem

In a confluent and terminating rewrite system unique normal forms exist.

Proof.

By termination, we can rewrite w to an irreducible string w'.

Now consider any irreducible descendant w'' of w. If $w' \neq w''$, then at least one of them can be rewritten by confluence, contradicting irreducibility.

□

The order in which we apply the reductions is irrelevant in the sense that we always arrive at the unique normal form. Note, though, that the derivation length may depend on our choices.

Leftmost Derivations

Suppose we have a confluent and terminating system. We can simplify the reduction to normal form by searching for a leftmost derivation: given w, find x, u and y such that

$$w = xuy \qquad |x| \text{ minimal}, |u| \text{ minimal}, u \to v \in \mathcal{P}$$

This sort of reduction can easily be handled by a finite state machine (a transducer). The whole reduction is just the transitive closure of a rational relation.

But note that there is no general bound on the number of steps needed to get to normal form.

Definition

A predicate ϕ on $\Sigma^\star$ is $\mathcal{P}$-inductive iff for all $x \in \Sigma^\star$

$$\forall\, z \left(x \to z \Rightarrow \phi(z)\right) \Rightarrow \phi(x)$$

Predicate ϕ is universal if it holds for all $x \in \Sigma^\star$.

$\mathcal{P}$ is well-founded iff every $\mathcal{P}$-inductive predicates is already universal.

In other words, in a well-founded r/w-system we have a method of induction familiar from $\mathbb{N}$, or, more generally, partial well-orders.

The following lemma justifies this terminology.

Lemma

A r/w-system is $\mathcal{P}$ is terminating iff it is well-founded.

Proof.

For LtR, let $S = \{ x \in \Sigma^\star \mid \phi(x) \}$ and assume $w \in \Sigma^\star - S$.

Since ϕ is inductive, w must have an immediate descendant $w' \notin S$. Continuing inductively, we get a chain $w \to w_1 \to w_2 \to \ldots$ in the complement of S.

By termination, the chain is finite, so some w_n has no descendants. But then $w_n \in S$, contradiction.

For RtL, consider the predicate $\phi(x)$ defined as: there is no infinite chain starting at x. Obviously, ϕ is $\mathcal{P}$-inductive, and hence universal by assumption.

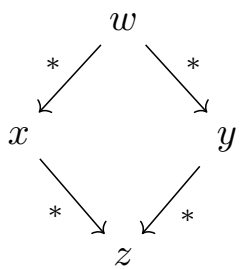
□

Theorem

A terminating r/w-system is confluent iff it is locally confluent.

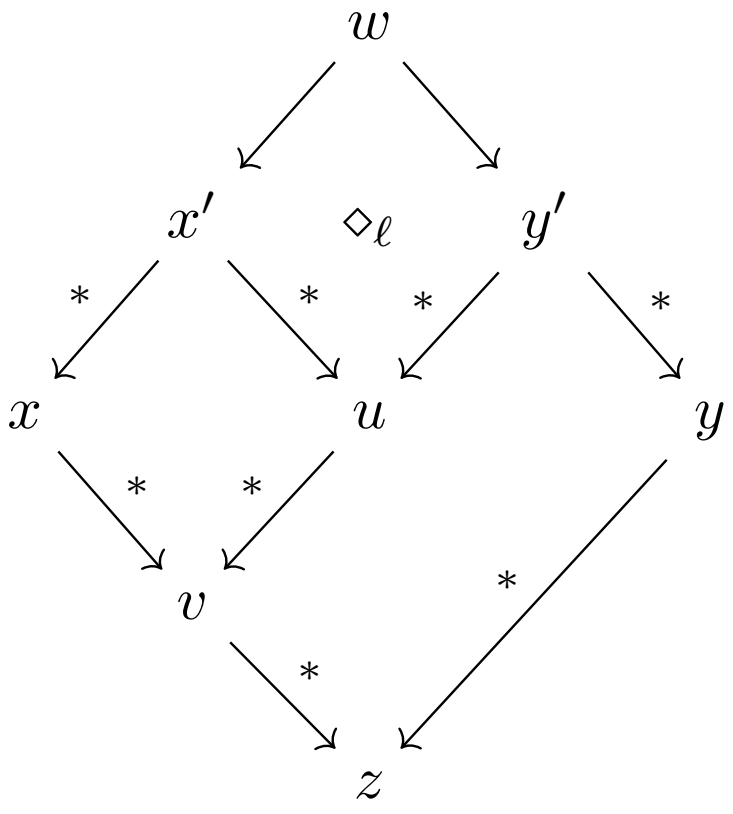
Proof. Confluence implies local confluence, so assume our system is terminating and locally confluent.

Consider the predicate $\phi(w)$: for all x and y such that $w \xrightarrow{*} x$, $w \xrightarrow{*} y$ implies $x \downarrow y$. In other words, w works on top of the confluence diamond:



By termination, it suffices to show that ϕ is inductive.

Proof



The upper diamond is local confluence, the others are plain confluence and justified by induction.

□

Lemma (Kőnig's Infinity Lemma)

Suppose T is an infinite but finitely branching tree. Then T contains an infinite branch.

Proof. Let x_0 be the root of T and $y_1, \ldots, y_k$ all its immediate descendants. Since the subtree with root x_0 is infinite, there must be at least one y_i so that the subtree with root y_i is also infinite. Set $x_1 = y_i$, and continue inductively.

□

Corollary

In a terminating r/w-system, the set of descendants of any string is finite.

Definition

A partial order $>$ on $\Sigma^\star$ is admissible iff

$$u > v \quad \text{implies} \quad \forall\, x, y \left(xuy > xvy \right)$$

A reduction order for $\mathcal{P}$ is any admissible well-order $>$ such that

$$u \to v \in \mathcal{P} \quad \text{implies} \quad u > v$$

If $>$ is admissible and all rules in $\mathcal{P}$ are ordered wrto $>$, then

$$x \xrightarrow{+} y \quad \text{implies} \quad x > y$$

Thus, to ensure termination it suffices to find a suitable reduction order for $\mathcal{P}$.

Lemma

The following are equivalent:

1. $\mathcal{P}$ *is Noetherian.*
2. *There is a reduction order for* $\mathcal{P}$*.*

Proof.

If $\mathcal{P}$ is Noetherian we can define

$$x > y \iff x \xrightarrow{+} y$$

For the opposite direction, recall that $x \xrightarrow{+} y \Rightarrow x > y$. Since $>$ is a well-order there cannot be infinite reduction chains.

□

Length Order: $x > y$ if $|x| > |y|$.

Length-Lex Order: $x >_{\mathrm{ll}} y$ if $|x| > |y|$ or $|x| = |y|$ and $x >_{\mathrm{lex}} y$.

Weight Order: Let $f : \Sigma \to \mathbb{N}$ and extend f to $\Sigma^\star$ as a monoid homomorphism. $x >_f y$ if $f(x) > f(y)$.

All these orders are admissible and well-founded. The last two are total, the first is only partial.

Note that for non-trivial alphabets ordinary lexicographic order $>_{\mathrm{lex}}$ is not well-founded (it is total and admissible, though).

Note that in algebraic manipulations one often would like to use associativity and commutativity and rewrite terms according to

$$a * (b * c) \to (a * b) * c$$
$$a * b \to b * a$$

Alas, just as often one would like to rewrite in the opposite direction: equations in algebra really do not carry any direction.

That causes huge problems with termination: a simple-minded algorithm could just go back and forth between $a * b$ and $b * a$ forever.

And restricting rules in any obvious way is also problematic since we might then miss out on important simplifications. There are special A/C rewrite sytems to deal with this.

Early in the 20th century Axel Thue was interested in the following question.

> Given a set of objects, and a collection of "rules" (transformations that turn objects into objects). Can a given object x be transformed into object y? Is there some third object z such that both x and y can be transformed into z?

Thue's questions make sense for lots of combinatorial structures such as terms, trees or graphs, but we will only consider the special case when the objects are all strings.

There is a nice algebraic interpretation:

Problem:	**Word Problem for Monoids**
Instance:	A monoid M and elements x and y of M.
Question:	Is $x = y$ in M?

Of course, for this to make any computational sense the monoid M must be given in some useful form: its elements must have a good finitary description. Also, we must be able to handle the algebra in M effectively: at the very least we must be able to perform multiplication and recognize the unit element.

The problem then is to determine equality of elements given by some sufficiently nice representation.

A good way to produce such effective descriptions is to start with the free monoid $\Sigma^\star$ over a (not necessarily finite) set Σ: this is just the collection of all finite words over Σ.

The operation is concatenation and the unit element is the empty word ε.

For example, $\Sigma = \{a\}$ produces a monoid isomorphic to $\langle \mathbb{N}, +, 0 \rangle$.

For better examples, we need to be able to "identify" certain words as denoting the same element in the monoid.

For example, let $\Sigma = \{a, b\}$ and let's insist that

$$ab = ba$$

Then we get a monoid isomorphic to the standard free Abelian monoid $\langle \mathbb{N}^2, +, 0 \rangle$.

Lemma

The Thue equivalence $\overset{*}{\longleftrightarrow}_{\mathcal{P}}$ is a congruence on $\Sigma^\star$.

Proof.

It is clear that $\overset{*}{\longleftrightarrow}_{\mathcal{P}}$ is in fact an equivalence relation.

So assume $x \overset{*}{\longleftrightarrow}_{\mathcal{P}} y$ and $u \overset{*}{\longleftrightarrow}_{\mathcal{P}} v$. Then $xu \overset{*}{\longleftrightarrow}_{\mathcal{P}} yv$ since the substitutions can be applied to the two parts of the word separately.

□

But then we can form the quotient monoid

$$M_{\mathcal{P}} = \Sigma^\star / \overset{*}{\longleftrightarrow}_{\mathcal{P}}$$

More precisely, since $\overset{*}{\longleftrightarrow}_{\mathcal{P}}$ is an equivalence relation, we can consider the equivalence classes $[x]$ of $x \in \Sigma^\star$.

Since the relation is a congruence, we can define a multiplication on these equivalence classes in the obvious way:

$$[x] \cdot [y] = [xy]$$

The result is a monoid $M_{\mathcal{P}} = \Sigma^\star / \overset{*}{\longleftrightarrow}_{\mathcal{P}}$.

In this context Σ is called a set of generators for $M_{\mathcal{P}}$.

It is easy to see that every monoid M can be written in the form $M_{\mathcal{P}}$ (just use M as Σ and let $\mathcal{P}$ be all valid equations in M).

Of course, this is not particularly exciting. The interesting case is when Σ is finite and $\mathcal{P}$ is finite, in which case we speak of a finite representation.

Given a finite representation we can easily check $x \leftrightarrow_{\mathcal{P}} y$ by table lookup.

But how about the word problem in $M_{\mathcal{P}}$, which comes down to checking $x \overset{*}{\longleftrightarrow}_{\mathcal{P}} y$?

For chains of substitutions of bounded length the problem is primitive recursive, but what if there is no bound?

Let $\Sigma = \{a, \bar{a}, b, \bar{b}\}$ and define $\mathcal{P}$ to be:

$$a\bar{a} \to \varepsilon \quad \bar{a}a \to \varepsilon \quad b\bar{b} \to \varepsilon \quad \bar{b}b \to \varepsilon$$
$$ba \to ab \quad b\bar{a} \to \bar{a}b \quad \bar{b}a \to a\bar{b} \quad \bar{b}\bar{a} \to \bar{a}\bar{b}$$

Claim: Using only the first 4 rules we can reduce any word w to the form

$$W = s_1^{k_1} t_1^{\ell_1} s_2^{k_2} t_2^{\ell_2} \ldots s_r^{k_r} t_r^{\ell_r}$$

where all the exponents are positive, except possibly k_1 and ℓ_r, $r \geq 0$; further $s_i \in \{a, \bar{a}\}$ and $t_i \in \{b, \bar{b}\}$. In fact, W is uniquely determined by w.

This is straightforward but tedious to show by induction on the length of w.

To simplify notation, define for $i \in \mathbb{Z}$

$$a^{(i)} = \begin{cases} a^i & \text{if } i \geq 0 \\ \bar{a}^{-i} & \text{otherwise.} \end{cases}$$

and likewise for b. Then

$$W = a^{(k_1)} b^{(\ell_1)} a^{(k_2)} b^{(\ell_2)} \dots a^{(k_r)} b^{(\ell_r)}$$

where the exponents are non-zero integers (except perhaps for k_1 and ℓ_r). Let us refer to the number r of ab-blocks in W as the width of W.

Claim: We can reduce the width of W to 1.

$$\begin{aligned} W &= a^{(k_1)} b^{(\ell_1)} a^{(k_2)} b^{(\ell_2)}\ U \\ &\xrightarrow{*} a^{(k_1)} a^{(k_2)} b^{(\ell_1)} b^{(\ell_2)}\ U \\ &\xrightarrow{*} a^{(k_1+k_2)} b^{(\ell_1+\ell_2)}\ U \end{aligned}$$

We have seen that the Thue congruence defined by $\mathcal{P}$ is a monoid congruence, so we can form the quotient structure

$$\Sigma^\star / \overset{*}{\longleftrightarrow}_{\mathcal{P}}$$

A priori, this is just a monoid, but a closer look reveals that it is isomorphic to the Abelian group $\mathbb{Z} \times \mathbb{Z}$: the first 4 rules ensure that $\bar{a}$ is the inverse of a, and likewise for $\bar{b}$. If we were to use only the first 4 rules in $\mathcal{P}$, we would obtain the free group on two generators a and b.

The last 4 rules provide the commutation required to get to $\mathbb{Z}^2$.

The group operation is concatenation followed by a reduction to normal form:

$$a^{(k)}b^{(\ell)} \cdot a^{(k')}b^{(\ell')} \overset{*}{\longleftrightarrow} a^{(k+k')}b^{(\ell+\ell')}$$

This shows clearly the isomorphism between the quotient structure and $\mathbb{Z}^2$.

Representing groups as quotient monoids is a standard technique in algebra.

In conjunction with r/w-systems we obtain an effective representation (efficiency is another matter, alas).

Theorem (E. Post, A.A. Markov 1947)

The word problem for finitely presented monoids is undecidable.

This theorem is often considered to be the first undecidability result in pure mathematics (as opposed to inside logic).

A simple example due to G. Makanin looks like this:

$$\begin{aligned}\mathcal{P}\colon\quad & ccbb \to bbcc, bcccbb \to cbbbcc, accbb \to bba,\\ & abcccbb \to cbba, bbccbbbbcc \to bbccbbbbcca\end{aligned}$$

There is a corresponding result for groups (Novikov-Boone 1955), but that is significantly harder to prove. For commutative monoids the word problem is decidable.

We would like to have an algorithm for the following problem:

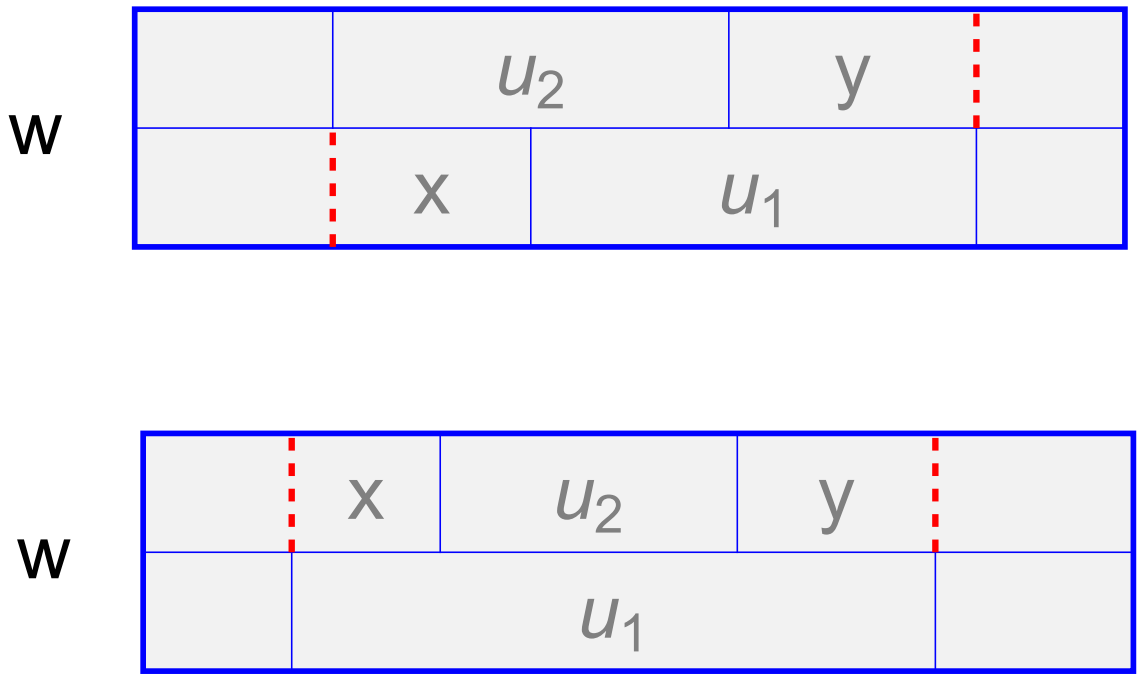
Problem:	**Confluence**
Instance:	A terminating r/w-system $\mathcal{P}$.
Question:	Is $\mathcal{P}$ confluent?

Since we assume that $\mathcal{P}$ is terminating it suffices to check for local confluence. So assume $w \to x$ and $w \to y$.

If the two handles used do not overlap, it is straightforward to obtain confluence: just take the other reduction next.

$$w = \alpha\, u_1\, \beta\, u_2\, \gamma$$

The real problem is to deal with overlapping handles: the first reduction step interferes with the second.



Collisions of type overlap and factor.

Let $u_1 \to v_1, u_2 \to v_2 \in \mathcal{P}$ be two rules (possibly the same).

The set of critical pairs for $\pi_1 : u_1 \to v_1, \pi_2 : u_2 \to v_2$ is defined as follows:

$$\begin{aligned}\mathsf{CP}(\pi_1, \pi_2) = \\ &\{ (xv_1, v_2y) \mid xu_1 = u_2y, |x| < |u_2| \} \cup \\ &\{ (v_1, xv_2y) \mid u_1 = xu_2y \}\end{aligned}$$

We say that a critical pair (z_1, z_2) is confluent iff $z_1 \downarrow z_2$.

Proposition

Let $\mathcal{P}$ be terminating.
$\mathcal{P}$ is confluent iff all critical pairs are confluent.

We can test confluence of a critical pair by computing the normal form of each component, say, by using a leftmost derivation. As already mentioned, there is no general bound on the lengths of these derivations.

For two rules π_1 and π_2 such that $|u_1| \geq |u_2|$ there are at most $|u_1|$ critical pairs. In fact, the construction of all critical pairs takes time quartic in $\|\mathcal{P}\|$.

Theorem

The Confluence Problem for terminating r/w-systems is decidable.

Let $\mathcal{P}$ be terminating with a reduction order $>$ and suppose our confluence test turns up a critical pair (z_0, z_1) that fail to be confluent. Say, $\widehat{z_i}$ is a normal form of z_i.

We could try to make $\mathcal{P}$ confluent by adding a rule $\widehat{z_i} \to \widehat{z_{1-i}}$, ordered according to $>$ (so as to preserve termination).

Iterating this method, we obtain an increasing sequence of r/w-systems that are equivalent in the sense that they generate the same Thue congruence. We stop as soon as we reach a fixed point.

// Knuth-Bendix

$R = \{\, u \to v \mid u > v, u \to v \in \mathcal{P} \vee u \to v \in \mathcal{P} \,\}$

repeat

  $R' = \emptyset$

  forall critical pairs (z_1, z_2) **do**

    compute normal forms $\widehat{z_1}$, $\widehat{z_2}$ wrto R

    if $\widehat{z_1} > \widehat{z_2}$ **then** add $\widehat{z_1} \to \widehat{z_2}$ to R'

    if $\widehat{z_2} > \widehat{z_1}$ **then** add $\widehat{z_2} \to \widehat{z_1}$ to R'

  od

  if $R = R' \cup R$

until $R' == \emptyset$

return R

Dire Warning: The completion process may not terminate.

Write R_k for the rule set R at the end of round k.
If the algorithm terminates we have output $\widehat{\mathcal{P}} = R_{k+1} = R_k$.

Without termination we still get "output" $\widehat{\mathcal{P}} = \bigcup R_k$, but this rule set is infinite (and recursively enumerable, for what that's worth).

Whether the completion algorithm terminates or not, we still have

1. $\widehat{\mathcal{P}}$ is terminating and confluent.
2. $\widehat{\mathcal{P}}$ and $\mathcal{P}$ are equivalent.
3. $>$ is a reduction order for $\widehat{\mathcal{P}}$.
4. $\mathsf{Irr}(\widehat{\mathcal{P}}) = \{\, x \in \Sigma^\star \mid \forall\, y \left(x \overset{*}{\longleftrightarrow}_{\mathcal{P}} y \Rightarrow x = y \vee y > x \right) \}$

Moreover, the procedure is optimal in the sense that it will terminate in finitely many steps iff there is a finite, confluent r/w-system equivalent to $\mathcal{P}$ that has $>$ as a reduction order.

Example: Commuting Group

Recall the example of the algebraic r/w-system $\mathcal{P}$ from above that defines $\mathbb{Z}^2$. This time, order the alphabet as in $\Sigma = \{a < \bar{a} < b < \bar{b}\}$.

$$a\bar{a} \to \varepsilon \quad \bar{a}a \to \varepsilon \quad b\bar{b} \to \varepsilon \quad \bar{b}b \to \varepsilon$$
$$ba \to ab \quad b\bar{a} \to \bar{a}b \quad \bar{b}a \to a\bar{b} \quad \bar{b}\bar{a} \to \bar{a}\bar{b}$$

It is easy to check that length-lex order is a reduction order for $\mathcal{P}$. Hence $R = \mathcal{P}$ at the beginning of the first round.

Next we have to identity all critical pairs. Since all LHSs have length 2 and are distinct, there are no factor-type pairs.

For overlap-type pairs we have $xu_1 = u_2y$ and $|x| < |u_2| = 2$ implies that $x, y \in \Sigma$. Hence we are really looking for LHSs u_1 and u_2 such that $u_{11} = u_{22}$.

This time, there are several possibilities.

Here is one typical example:

$$b \mid a\bar{a} = ba \mid \bar{a} \quad \rightsquigarrow \quad (b, ab\bar{a})$$

Since $ab\bar{a} \to ba\bar{a} \to b\varepsilon = b$, no new rule is added.

This shows that $\mathcal{P}$ is already confluent and the algorithm returns $\widehat{\mathcal{P}} = \mathcal{P}$ in one round.

Different Order

Suppose we change the order to $\Sigma = \{\bar{b} < a < \bar{a} < b\}$.

This forces us to flip the last two rules for R in round 1: $a\bar{b} \to \bar{b}a$, $\bar{a}\bar{b} \to \bar{b}\bar{a}$. We get critical pairs $(a, \bar{b}ab)$ from $\bar{b}ba$ and $(\bar{a}, \bar{b}\bar{a}b)$ from $\bar{b}b\bar{a}$, leading to new rules $\bar{b}ab \to a$ and $\bar{b}\bar{a}b \to \bar{a}$.

In round 2 we get additional rules $\bar{b}a^2b \to a^2$ and $\bar{b}\bar{a}^2b \to \bar{a}^2$.

In general, we pick up new rules $\bar{b}a^kb \to a^k$ and $\bar{b}\bar{a}^kb \to \bar{a}^k$ in the kth round.

The algorithm does not terminate.

Emil Post analyzed Russell and Whitehead's *Principia Mathematica* with a view towards building simpler systems that would still have the same proof power.

Somewhat surprisingly, Post managed to interpret Principia as an exercise in word processing[†]: everything comes down to a purely mechanical manipulation of strings, semantics being entirely optional. The key point is that the manipulations require no insights whatsoever, they are all trivially algorithmic.

[†]Very much in the spirit of Hilbert: math can be construed as a game in which ink marks are being pushed around on paper.

Let Σ and $\mathcal{X}$ be two finite alphabets, referred to as terminals and variables, respectively.

A Post production rule is an expression

$$W_1, W_2, \ldots, W_k \Rightarrow U$$

where $W_i, U \in (\Sigma \cup \mathcal{X})^\star$.

A Post canonical system (PCS) $\mathcal{P}$ consists of axioms (a finite set of words over Σ) and a finite collection of production rules.

The language $\mathcal{L}(\mathcal{P}) \subseteq \Sigma^\star$ of a PCS $\mathcal{P}$ is define inductively:

- All axioms are in $\mathcal{L}(\mathcal{P})$.
- Let σ be a binding from $\mathcal{X}$ to $\Sigma^\star$ and $W_1, W_2, \ldots, W_k \Rightarrow U$ a production. If all $W_i[\sigma]$ are in $\mathcal{L}(\mathcal{P})$, then so is $U[\sigma]$.

It is straightforward to define a corresponding notion of derivation, either as a linear sequence, or as a tree.

In light of Post's "analyze Principia" project, the language of a PCS is also called its set of theorems.

In the following, $\Sigma = \{a, b\}$ and the only axiom is ε.

Production rules for "balanced parentheses":

$$X \Rightarrow aXb$$
$$X, Y \Rightarrow XY$$

Production rules for "same number of as and bs":

$$XYZ \Rightarrow XaYbZ$$
$$XYZ \Rightarrow XbYaZ$$

Here is a sample derivation of $abbaab$ in the second PCS.

	X	Y	Z	rule
ε	$-$	$-$	$-$	ax.
ab	ε	ε	ε	1
$abab$	a	ε	b	2
$abbaab$	ab	ε	ab	2

Exercise

Show that this PCS works as advertised: it's language consists of all strings over Σ with the same number of as and bs.

It is clear that there is an algorithm to test whether an alleged derivation is actually correct according to the rules of a particular PCS (even if we are only given the first column).

But beware: it may well be undecidable whether $w \in L$: a priori, the only way to tackle this question is to search systematically for a corresponding derivation.

Alas, this approach is really not very helpful:

- If $w \in L$, our search will terminate with answer YES.
- If $w \notin L$, our search will go on forever.

Hence the theorems of a PCS are semidecidable, but may well fail to be decidable.

Post managed to prove a surprising normal form theorem for his systems.

Theorem (Post 1943)

All the productions of a PCS can be written in the form

$$uX \Rightarrow Xv$$

More precisely, given a PCS $\mathcal{P}$ one can effectively construct a new system $\mathcal{P}'$, possibly over a larger alphabet, using only this type of productions, so that $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}') \cap \Sigma^\star$.

Note that derivations in normal form system are less complicated, it is easier to find the right binding for the variable.

Here is a rather useful way of expressing Post normal form.

Definition

A Post d-tag system consists of an alphabet Σ, a deletion number d and a map $P : \Sigma \to \Sigma^\star$.

The system defines the following rewrite rules

$$a_1\, a_2 \ldots a_d\, x \to x\, P(a_1)$$

Thus, the rewriting here only occurs at the end of the string:

- First remove the first d letters, then
- append the suffix $P(a)$ where a was the first letter.

So this is much like standard queue operations.

Example 2-Tag

The alphabet is $\Sigma = \{a, b, c\}$ and the rules are

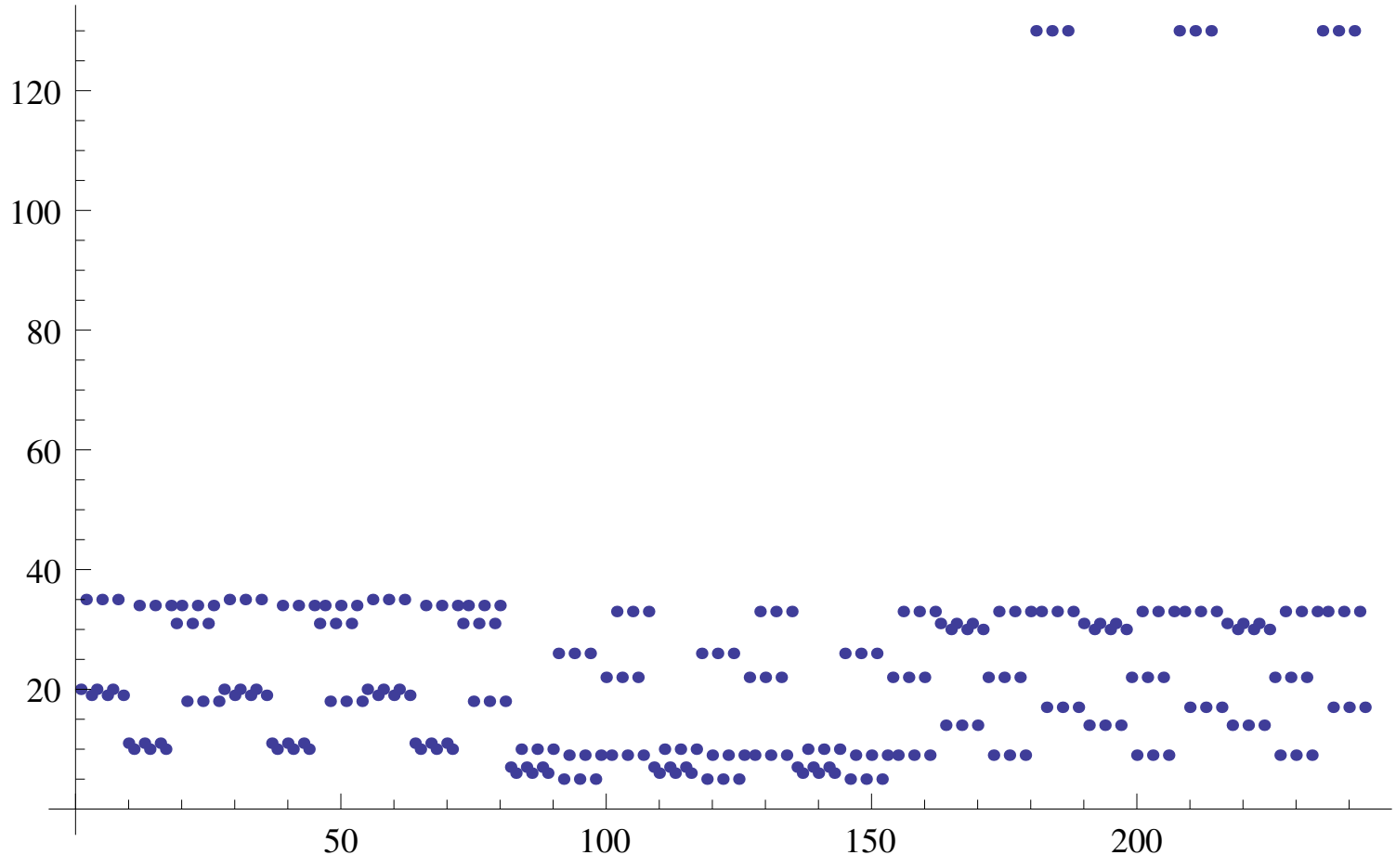
$$a \to bc \quad b \to a \quad c \to aaa$$

For example, we have the following, rather circuitous, derivation from aaa to a:

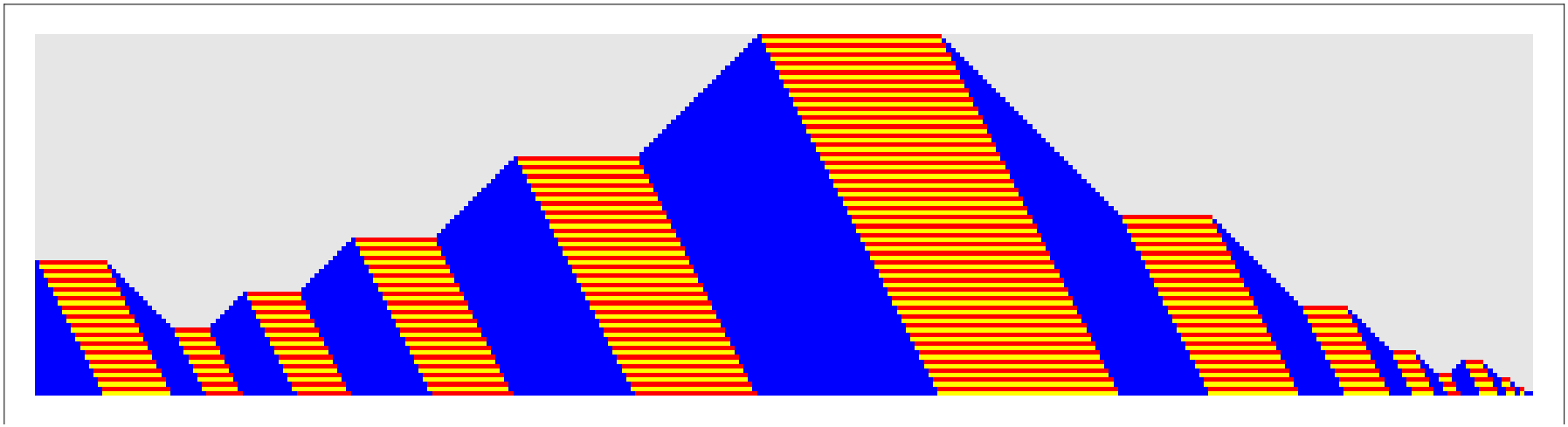
$$aaa, abc, cbc, caaa, aaaaa, aaabc, abcbc, cbcbc, cbcaaa, caaaaaa, aaaaaaaa,$$
$$aaaaaabc, aaaabcbc, aabcbcbc, bcbcbcbc, bcbcbca, bcbcaa, bcaaa, aaaa, aabc,$$
$$bcbc, bca, aa, bc, a$$

At this point the derivation ends since we have no 2 letters left to erase.

In this system, all initial words produce a terminating derivation.



The transients (leading to a fixed point), for all 243 words of length 5. Words are sorted in lexicographic order.



Here are the lengths of all words of the form $a^\star$ in the orbit of a^{30}:

$$30, 15, 23, 35, 53, 80, 40, 20, 10, 5, 8, 4, 2, 1$$

Yup, it's true . . .

Given any word w the tag system defines a sequence of words (w_i), the orbit of w.

Clearly there are the 3 basic possibilities:

- Halting:
  for some i, $|w_i| < d$ and the rewrite process stops.
- Periodic:
  $|w_i|$ remains bounded and the orbit is ultimately periodic.
- Unbounded:
  $|w_i|$ grows unboundedly.

Note that it is semidecidable whether w halts or becomes periodic.
Unboundedness is more complicated.

Theorem (Minsky 1961)

It is undecidable whether a word has an infinite orbit in a Post tag system.

The proof is quite hard and involves a reduction of ordinary Turing machines to "two-tape non-writing Turing machines" (two counters).

With more effort these can then be simulated by tag system.

These undecidability results are genuinely difficult, this is nothing like Halting.

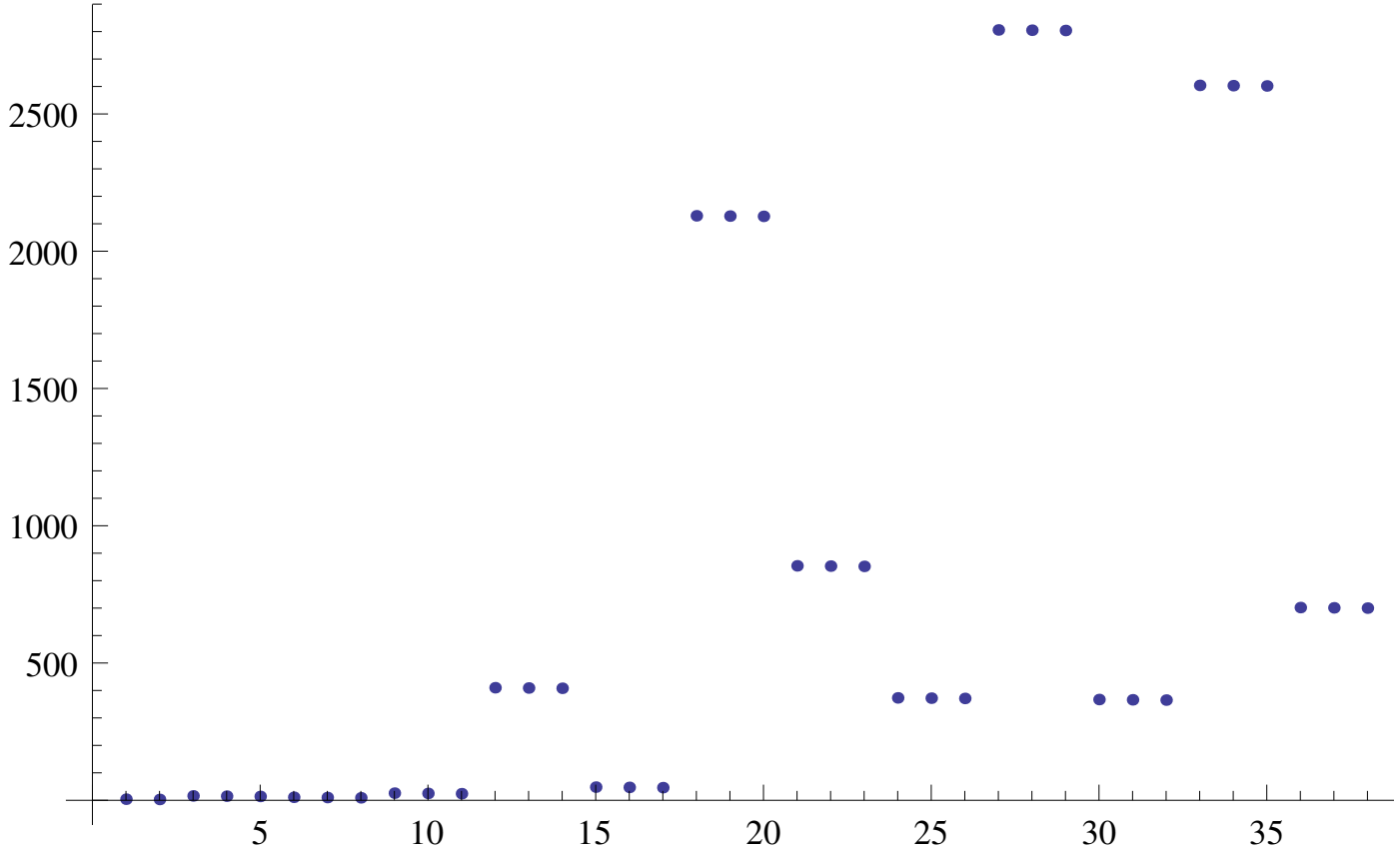
Post seems to have constructed a complete classification of all binary 2-tag systems (alas, he did not publish).

But for the $d = 3$ he was stumped by the following binary system:

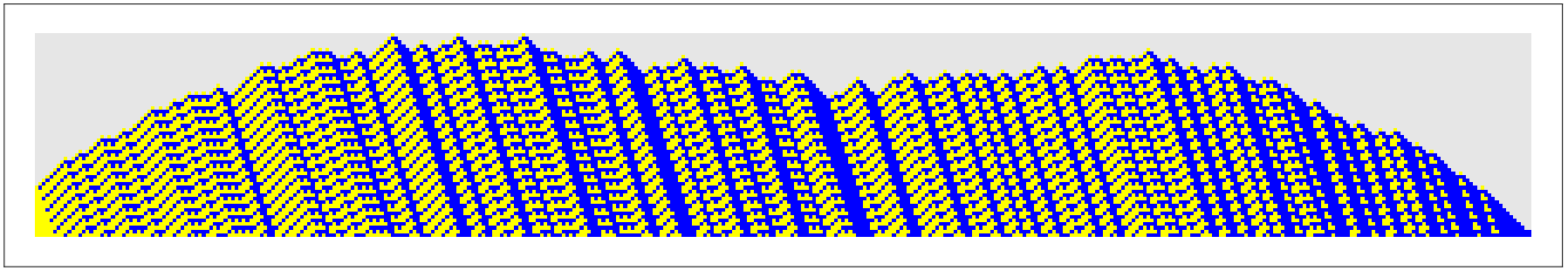
$$a \to aa \quad b \to bbab$$

At first glance, this system looks far to primitive to cause any serious problems: there are just two tiny rules; clearly, with a modest amount of effort, we should be able to completely understand this system. Right?

Alas . . .

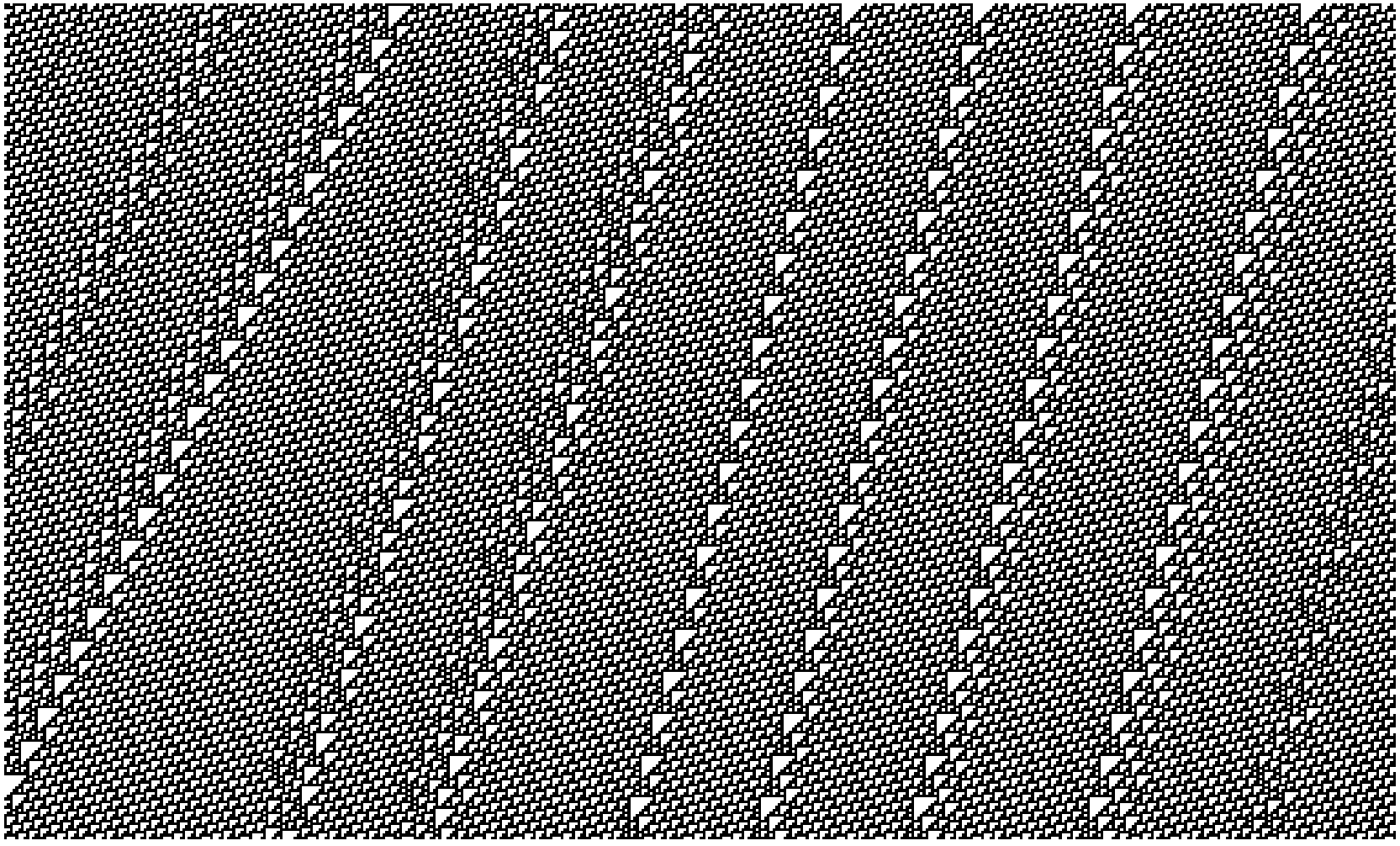


The length of all transients of words b^k, $k \leq 40$. The periods are 1, 2 and 6.



Lots of structure, but just enough chaos to stump everybody.

The proof for the computational universality of elementary cellular automaton 110 is based on a simulation of a type of tag system.



Post had originally hoped to obtain a good theory that would cover all formal systems (including e.g. Russel and Whitehead's *Principia Mathematica*).

His initial success was to show that they all could be construed as tag systems. But then he realized that even for $d = 3$ these systems are too difficult to deal with, a very unhappy experience.

> . . . the best I can say that I would have proved Gödel's theorem in 1921—had I been Gödel.
>
> E. Post, 1938

We will take a look at a type of rewrite system that seems appealing from the algorithmic perspective: we only use prefixes as handles (rather than factors as in the general case). So, finding a handle is particularly simple here.

Alas, we will see that this is one step too far: we wind up essentially with just regular languages.

Still, this is useful to analyze finite state machines, and even produces interesting results in group theory.

Definition

A prefix rewrite system (PRWS) is a pair

$$\mathcal{R} = \langle \Gamma, \mathcal{P} \rangle$$

where $\Gamma = \Sigma \cup \mathcal{X}$ is an alphabet and $\mathcal{P} \subseteq \Gamma^\star \times \Gamma^\star$ is a finite set of productions. Σ is the set of terminals and $\mathcal{X}$ is the set of non-terminals.

Non-terminals are also referred to as auxiliary symbols or syntactic variables, the latter terminology is used in particular in conjunction with formal grammars, a special type of rewrite system.

We modify the one-step relation as follows:

$$ux \to_{\mathcal{P}} vx \iff u \to v \text{ in } \mathcal{P}$$

For languages, the dichotomy between analysis and synthesis takes the following form:

Recognition Develop an algorithm that, given a word, determines whether it belongs to the language.

Generation Develop a method that allows one to generate all words in a language in some systematic fashion.

From a computability perspective, recognition requires decidability, but generation only requires recursive enumerability. Hence, some languages can be generated but not recognized.

However, for families of simple languages one would expect a connection between the two approaches.

In our setting, we can modify to the definition of the language associated with a PRWS in the following useful manner. Given $A, B \in \mathcal{X}$ let

$$\mathcal{L}(\mathcal{P}, A, B) = \{\, x \in \Sigma^\star \mid Ax \xrightarrow{*} B \,\}$$

We could also use sets of non-terminals. Note that the language is still a subset of $\Sigma^\star$, though non-terminals are used during the derivation. In this section we will focus on recognition, whence this definition.

Exercise

The $\{\, x \in \Sigma^\star \mid A \xrightarrow{*} Bx \,\}$ could be considered as the language generated by a PRWS system. What is the relationship to the recognition language?

We can constrain the form of our productions without affecting the language. In particular, it is easy to see that productions of the form

$$\begin{aligned} Aa &\to B && \text{contracting} \\ A &\to B && \text{neutral} \\ A &\to Bb && \text{expanding} \end{aligned}$$

suffice where $a, b \in \Sigma$—albeit at the cost of potentially increasing the number of non-terminals. We will call these systems Reduced PRWS. Note that in a RPRWS one only needs to consider derivations involving words of the form $\mathcal{X}\Sigma^\star$.

The next result is a bit more surprising:

Lemma

For every PRWS there is an equivalent reduced PRWS without expanding rules.

Proof

The new RPRWS $\mathcal{P}'$ uses the same alphabets, but the rules are changed according to

$$Aa \to_{\mathcal{P}'} B \qquad \text{where} \qquad Aa \xrightarrow{*}_{\mathcal{P}} B$$

$$A \to_{\mathcal{P}'} B \qquad \text{where} \qquad A \xrightarrow{*}_{\mathcal{P}} B$$

It follows that $x \xrightarrow{*}_{\mathcal{P}'} y$ implies $x \xrightarrow{*}_{\mathcal{P}} y$.

It remains to show that $x \to_{\mathcal{P}} y$ implies $x \xrightarrow{*}_{\mathcal{P}'} y$. To this end, consider a derivation in $\mathcal{P}$ of the form

$$A_0 x_0, A_1 x_1, \ldots, A_n$$

Let $x_0 = a_1 a_2 \ldots a_r \in \Sigma^\star$. For $i \in [r]$ define $\sigma(i)$ to be the maximum j such that $x_j = a_i a_{i+1} \ldots a_r$.

Hence, for all $i \in [r]$ the derivation has a step

$$A_{\ell(i)} a_i \dots a_r \to_{\mathcal{P}} A_{\ell(i)+1} a_{i+1} \dots a_r$$

But then we introduce the new productions

$$A_{\ell(i)} a_i \xrightarrow{*} A_{\ell(i)+1}$$

$$A_{\ell(i)} \xrightarrow{*} A_{\ell(i+1)}$$

and obtain an equivalent system without contractions. Here $A_{\ell(r+1)}$ is understood to be B.

□

Since we do not yet understand derivations in $\mathcal{P}$, the last result is somewhat unsatisfactory: the new rewrite system exists for reasons of finiteness, but we have no way to construct it.

Intuitively, there should be a reasonably simple way to bound the lengths of the derivations in $\mathcal{P}$ needed to produce $\mathcal{P}'$. In turn, to bound the lenghts of derivations, it suffices to bound the lengths of the words occurring in them.

To this end, define a block in a derivation

$$X_0x_0, X_1x_1, X_2x_2, \ldots, X_{n-1}x_{n-1}, X_nx_n$$

to be a maximal interval $[l, r]$ such that x_l appears as a suffix of x_i, $l \leq i \leq r$. It is clear that blocks are either nested or overlapping. Moreover, every terminal suffix of a word in a derivation uniquely determines a corresponding block.

A Bound

Let's say that a block $[l, r]$ is of type X, Y iff $X_l = X$ and $X_r = Y$.
So there are $|\mathcal{X}|^2$ different types of blocks.

By a shortest derivation we mean a derivation of minimum length given a fixed source and target.

Now consider a longest term Xx in a shortest derivation. The $|x|$-many suffixes of x give rise to a nested sequence of blocks. If that sequence is longer than $|\mathcal{X}|^2$, then two blocks of the same type must be nested. But then the derivation cannot be shortest, one can retain only the inner block and still have a valid derivation.

Hence $|\mathcal{X}|^2 + 1$ is an upper bound for the length of terms in any shortest derivation, yielding an obvious exponential bound for the length of such a derivation.

We will now construct an automaton $\mathcal{A}$ that accepts words recognized by an expansion-free reduced PRWS.

To this end, think of $\stackrel{*}{\longrightarrow}$ as a pre-order on $\mathcal{X}$. We use as state set of $\mathcal{A}$ the equivalence classes induced by this pre-order. Thus $P \subseteq \mathcal{X}$ is a state iff $\forall\, X, Y \in P (X \stackrel{*}{\longrightarrow} Y)$.

The transition function δ is given by $P \stackrel{a}{\to} P'$ iff $\exists\, X \in P, Y \in P\, (Xa \stackrel{*}{\longrightarrow} Y)$.

The following claim is easily established by induction:

Claim: $\delta(P, w) = P'$ iff $\exists\, X \in P, Y \in P'\, (Xw \stackrel{*}{\longrightarrow} Y)$.

Theorem

The language $\mathcal{L}(\mathcal{P}, A, B)$ of any prefix rewrite system $\mathcal{P}$ is regular.
A finite state machine for the language can be constructed from the system.

As an aside: The special case when the system is pure (i.e., $\mathcal{X} = \emptyset$) is actually of interest.
We need to adjust our definition of language slightly: the set $\{ x \mid w \xrightarrow{*} x \}$ of all words derivable from a fixed word w is regular.

Recall that a tree is a prefix-closed set $T \subseteq \Sigma^\star$. Define the frontier and the interior of $X \subseteq \Sigma^\star$ as follows.

$$\mathsf{fr}(X) = X \cdot \Sigma - X$$
$$\mathsf{int}(X) = \{\, x \mid \exists\, u \in X\, (x \sqsubset u) \,\}$$

A frontier is any set of the form $\mathsf{fr}(T)$ for some tree T.

Lemma

Let T be a tree and S a frontier. Then the following hold.

1. $\mathsf{int}(\mathsf{fr}(T)) = T$,
2. $\mathsf{fr}(\mathsf{int}(S)) = S$.

Let $A \subseteq \Sigma^\star$. The interior and exterior of A are defined as

$$\mathsf{int}(A) = \{\, x \mid \exists\, u \in A\, (x \sqsubset u) \,\}$$
$$\mathsf{ext}(A) = \{\, x \mid \exists\, u \in A\, (u \sqsubset x) \,\}$$

A is a frontier if A, $\mathsf{int}(A)$ and $\mathsf{ext}(A)$ form a partition of $\Sigma^\star$. If T is a tree, we refer to $T - \mathsf{int}(T)$ as the frontier of T.

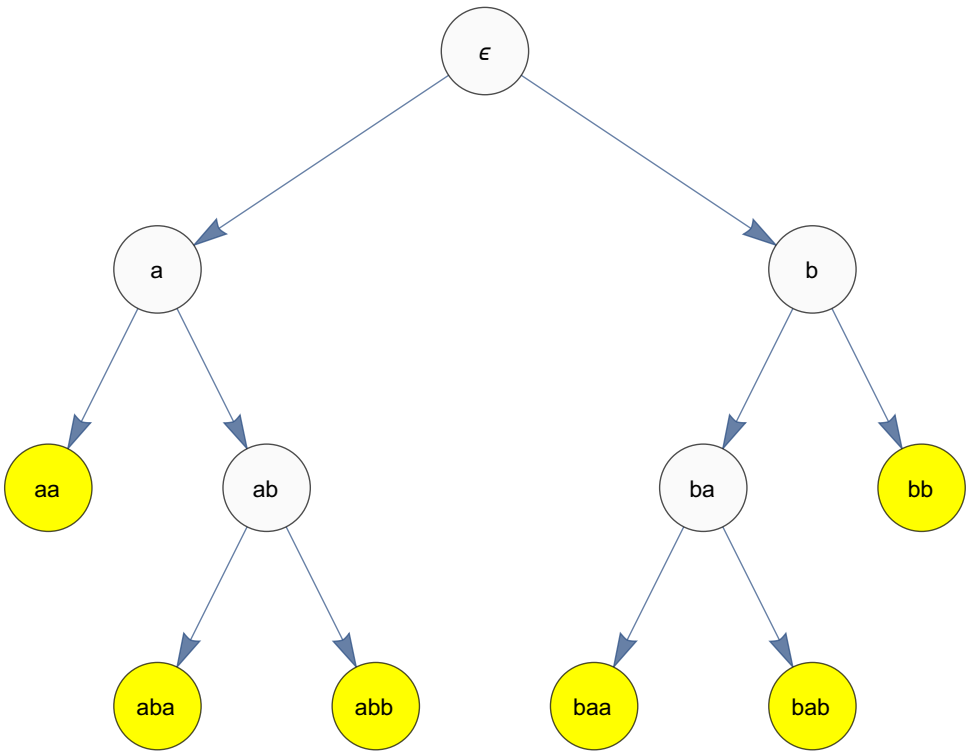
In the case of a single word w, $w \cup \mathsf{int}(w)$ is the branch leading to w, $w \cup \mathsf{ext}(w)$ is the subtree rooted at w.

Claim: For a frontier A, $x \neq y \in A$ implies that x and y are incomparable.

Claim: A is a frontier iff A is a maximal antichain.

Example: Tree plus Frontier

Here is a picture of the frontier $S = \{aa, bb, aba, abb, baa, bab\}$ of the tree $T = \{\varepsilon, a, b, ab, ba\}$, the interior of S.



Example: Even/Even

Let E be the language of all even/even words over $\Sigma = \{a, b\}$.

Here is a pure prefix-rewrite system that recognizes E in the sense that $x \xrightarrow{*} \varepsilon$ iff $x \in E$.

$$aa, bb \to \varepsilon$$
$$abb, bab \to a$$
$$baa, aba \to b$$

The words on the left-hand-side are the frontier of the tree $\{\varepsilon, a, b, ab, ba\}$, as on the last slide.

Also note that for any x there is a $v \in T$ such that $x \xrightarrow{*} v$.

The question arises whether the even/even example generalizes to some other regular languages. First, we need to formalize our framework.

Suppose $\mathcal{P} = \{ u_i \to v_i \mid i \in [n] \}$ is a pure PRS. Call $\mathcal{P}$ a frontier PRS if the set $U = \{ u_i \mid i \in [n] \}$ of left-hand-sides is a frontier and $v_i \in \mathrm{int}(U)$.

A frontier PRS recognizes a language L if there is a set $U_0 \subseteq U$ such that $L = \{ x \mid x \xrightarrow{*} w \in U_0 \}$.

Thus the rewrite system from the last slide is a frontier PRS, and it recognizes the even/even language.

First, a little sanity check: in any frontier PRS, all words can be rewritten to a word in the corresponding tree.

Lemma

For any word x there exists $w \in \mathrm{int}(U)$ such that $x \xrightarrow{*} w$.

Proof. Again we only need to consider x not in the interior; for these words recall the distance $d(x) \geq 0$ defined to be $|z|$ where $x = u_i z$. Distance is uniquely determined since U is a frontier. Now $x \to v_i z$, so if $v_i z$ lies in the interior, we are done. Otherwise $d(v_i z) < d(x)$, done by induction. □

Lemma

A language recognized by a frontier PRS is regular.

Proof.

We can build a DFA $\mathcal{A}$ for the language of the PRS as follows. Let U be the frontier of left-hand-sides.

- States are $Q = \mathsf{int}(U)$, initial state is ε.
- The set of final states is U_0.
- The transition function is given by

$$\delta(x, a) = \begin{cases} xa & \text{if } xa \in Q \\ v & \text{if } u = xa \in U,\ u \to v \in \mathcal{P}. \end{cases}$$

Clearly computations in $\mathcal{A}$ correspond to derivations in $\mathcal{P}$. □

Lemma

For every regular language L, there is a frontier PRS recognizing L. The PRS can be constructed in polynomial time from a DFA for L.

Proof.

Think of $\Sigma^\star$ as the complete k-ary tree, rooted at ε. Given a DFA $\mathcal{A}$ for L, we can label the nodes in $\Sigma^\star$ by states: $\lambda(x) = q_0 \cdot x$. Call a subtree $T \subseteq \Sigma^\star$ saturated if $\lambda(\mathsf{fr}(T)) \subseteq \lambda(T)$.

Construct a saturated subtree T, let $U = \mathsf{fr}(T)$ and define a frontier PRS by setting $u \to v \in \mathcal{P} \iff \lambda(u) = \lambda(v)$ for $v \in T$.

U_0 is given by the nodes in T labeled by final states. □

The saturated trees from the last proof have the following characterization.

Lemma

Assume $\mathcal{A}$ is accessible. Then $T \subseteq \Sigma^\star$ is saturated iff $\lambda(T) = Q$.

Proof. It suffices to show the direction from left to right.

First, for x not in T, define the distance $d(x) \geq 0$ to be $|z|$ where $x = uz$, $u \in \mathsf{fr}(T)$. It is easy to see that $d(x)$ is well-defined.

Now let $p \in Q$ and assume for a contradiction that $p \notin T$. By assumption, p is not on the frontier either. By accessibility, let $p = \lambda(x)$ and $x = uz$ where $u \in \mathsf{fr}(T)$.

Then $p = \lambda(vz)$ where $v \in T$ and $\lambda(u) = \lambda(v)$. Done by induction on $d(x)$.

□

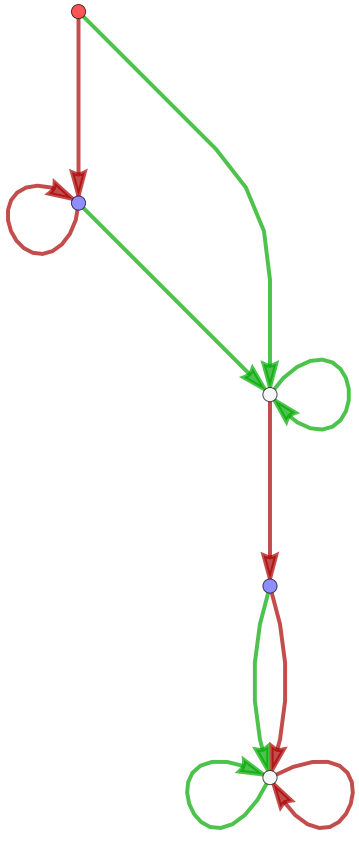
For the construction of a frontier PRS from a DFA, there are several interesting ways to build the required saturated tree.

Length-lex: traverse the tree in length-lex order (ordered BFS) and make sure that every label occurs exactly once in T.

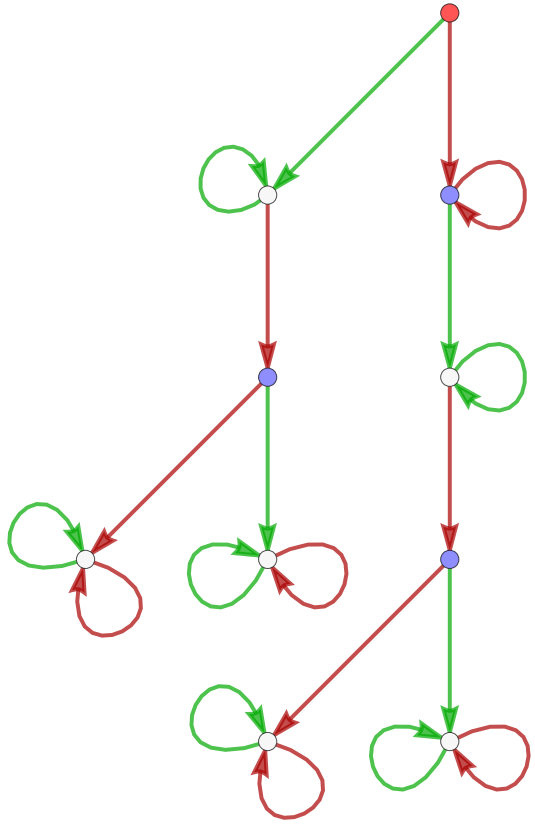
Periodic: make sure that x is on the frontier of T iff there is exactly one prefix $z \sqsubset x$ such that $\lambda(x) = \lambda(z)$.

Levels: Truncate $\Sigma^\star$ at level k for k sufficiently large.

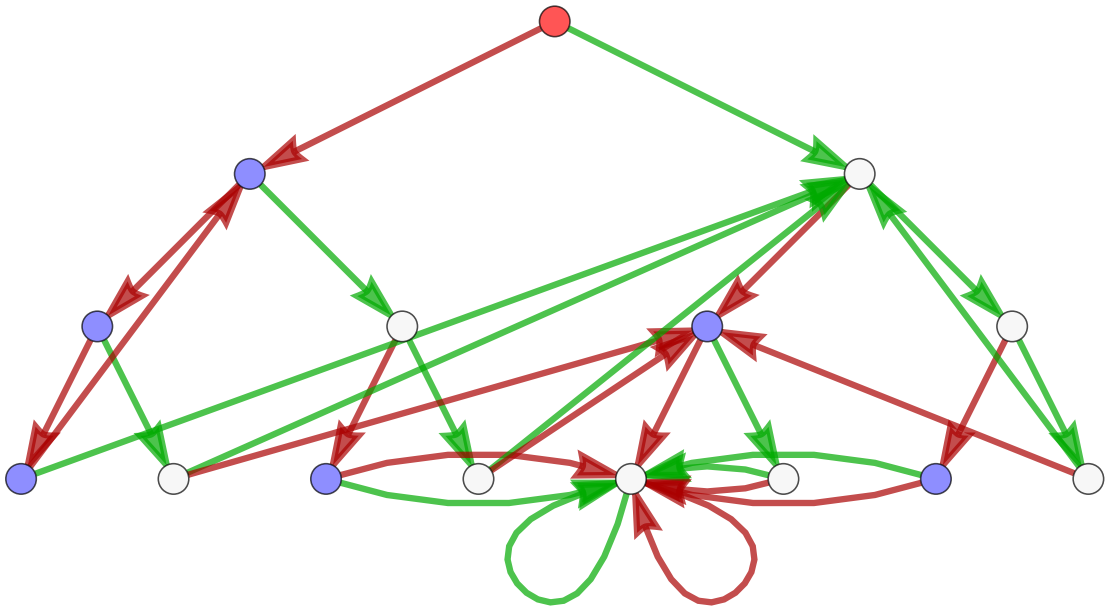
Since we can convert back to DFAs, we have a way to build machines that help to explain the structure of the underlying regular language.



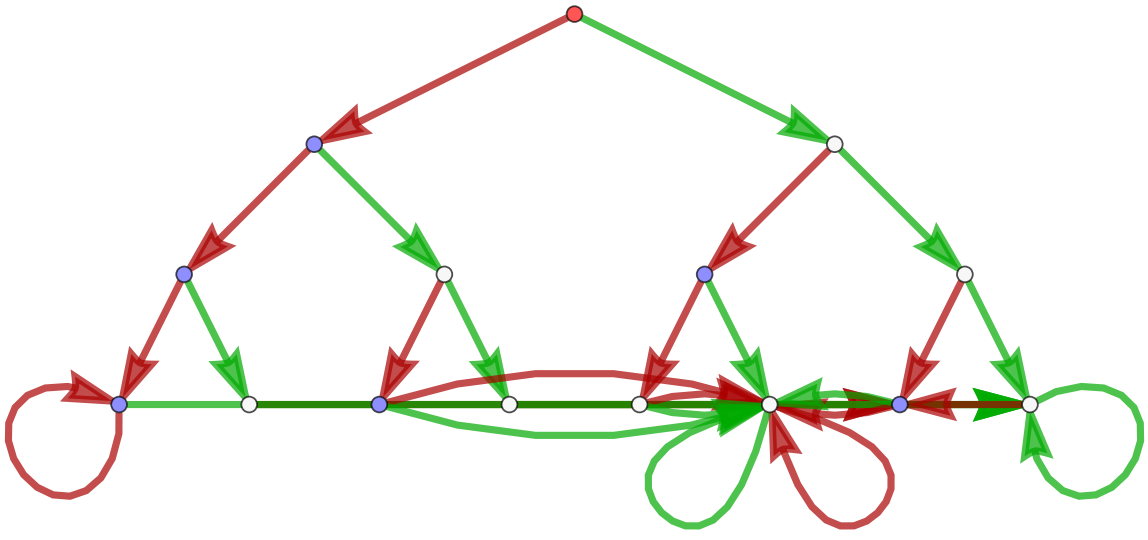
The minimal automaton, isomorphic to the length-lex frontier automaton.



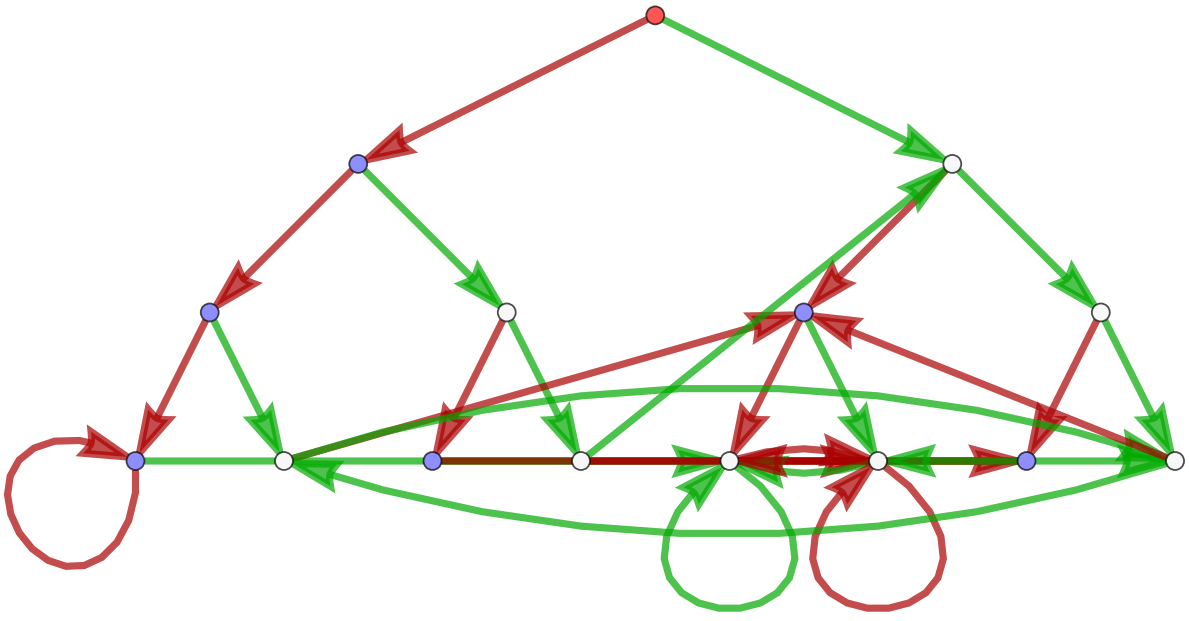
The periodic frontier automaton for $a^\star b^\star a$.



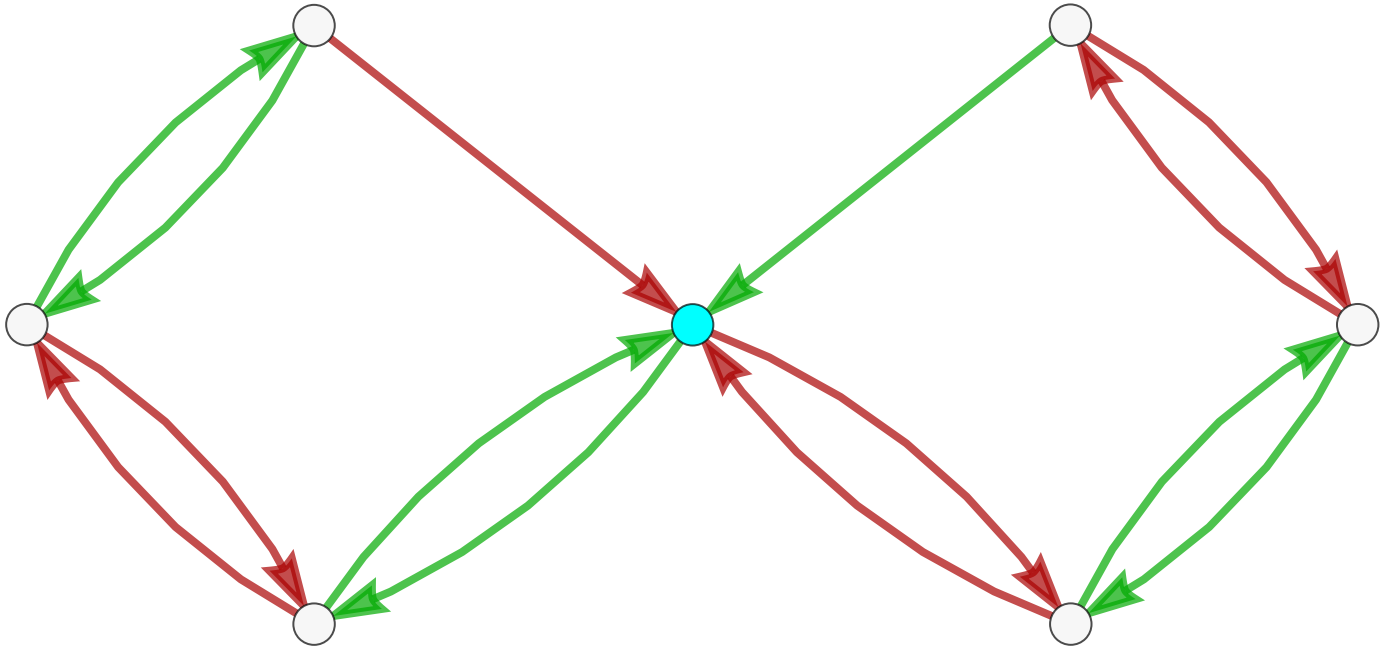
The frontier automaton for $a^\star b^\star a$ at level 4. Back-edges go to the first possible place.



The frontier automaton at level 4. Back-edges go to the last possible place.

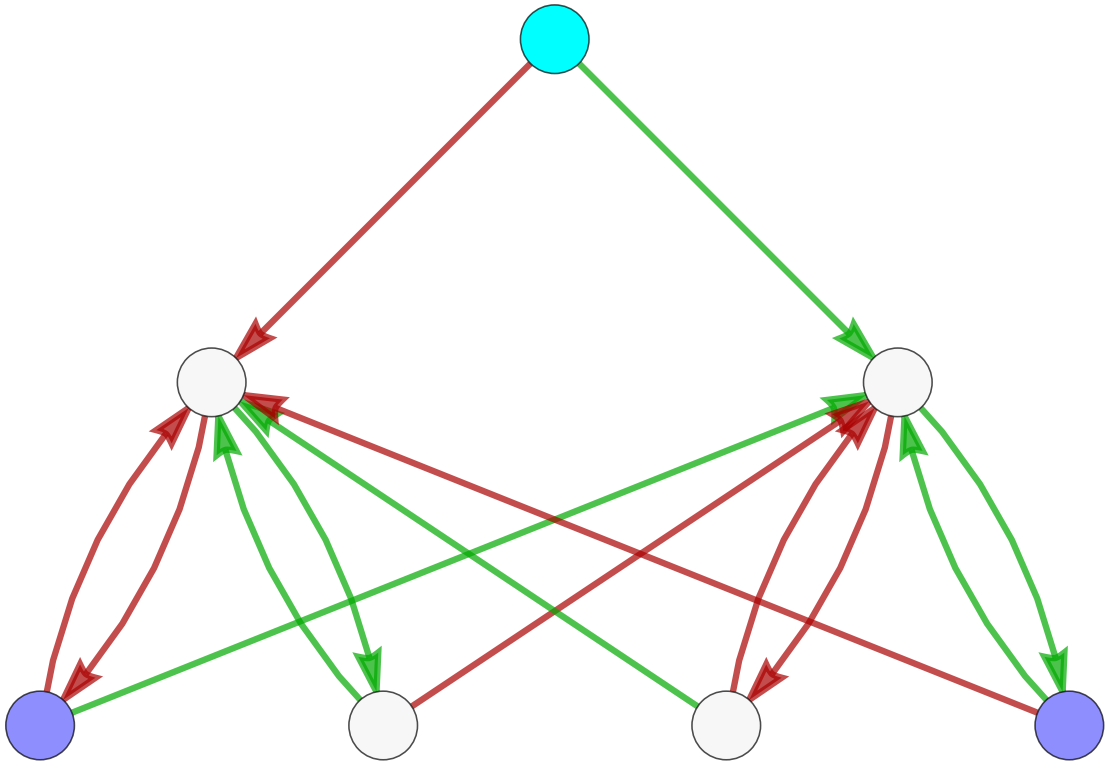


The frontier automaton for $a^\star b^\star a$ at level 4. Back-edges go to a random admissible place.



The periodic frontier automaton for even/even.

Make sure to figure out which states would merge under minimization.



The frontier automaton at level 3 for even/even. Note the symmetry around the middle axis (a convoluted way to prove that the language is invariant under swapping a and b).