CDM
Rewrite Systems

Klaus Sutner
Carnegie Mellon Universality

50-rewrite 2017/12/15 23:17

Where Are We? 3

We have two families of natural, grammar-based languages: context-free and context-sensitive. CFLs are important in the theory of programming languages, CSLs are important in complexity theory. Both have natural machine models (though LBAs are a bit beyond actual feasible computation).

It is natural to ask whether there are any other interesting types of grammars?

General Grammars 4

We will keep the basic framework:

Definition

A (formal) grammar is a quadruple

$$G = \langle V, \Sigma, \mathbb{P}, S \rangle$$

where V and Σ are disjoint alphabets, $S \in V$, and $\mathbb{P}$ is a finite set of productions or rules.

- the symbols of V are (syntactic) variables,
- the symbols of Σ are terminals,
- S is called the start symbol (or axiom).

Write $\Gamma = V \cup \Sigma$ for the complete alphabet of G.

General Productions 5

If we cling to our idea of replacing syntactic variables, the most general type of productions would look like so:

$$\pi : \alpha \to \beta \qquad \text{where } \alpha \in \Gamma^\star V \Gamma^\star, \beta \in \Gamma^\star$$

These grammars are called phrase-structure grammar.

So the only condition here is that α contains a syntactic variable; other than that, anything goes. In particular, α may well contain terminals–arguably, this is a bit weird.

Variants 6

A slight less chaotic type of grammar forbids the use of terminals on the left hand side of a production.

A type-0 grammar has productions of the form

$$\pi : \alpha \to \beta \qquad \text{where } \alpha \in V^+, \beta \in \Gamma^\star$$

Type-0 grammars and phrase-structure grammars are really the same: just replace terminal a on the LHS by A_a and add rules $A_a \to a$.

Then why introduce both?

The Chomsky Hierarchy (1956)

type 0	semidecidable
type 1	context-sensitive
type 2	context-free
type 3	regular

Note the date.

Phrase-Structure versus Semidecidable

Theorem

A language is phrase-structure iff it is semidecidable.

Proof.

Suppose G is a phrase-structure grammar for L.

Clearly we can construct a Turing machine that on input $x \in \Sigma^\star$ starts to enumerate all possible derivations in G (in some order, say, length-lex).

But then the language of G is recursively enumerable and thus semidecidable.

Proof, cont'd

For the opposite direction consider some deterministic one-tape Turing machine, say, $M = \langle\, Q, \Sigma, \Gamma, \delta, q_0, q_H \,\rangle$, that accepts L.

Let

$$V = \Sigma_\varepsilon \times \Gamma \cup Q \cup \{S, T, U\}$$

Here S, T and U are new symbols and everything is disjoint.

We may safely assume that all transitions are of the form $\delta(p, C) = (q, D, \pm 1)$ (i.e., the tape head always moves). Also, we may assume that the TM never moves to the left beyond the original input (i.e., uses a one-way infinite tape).

Batten down the hatches.

Productions

$$\begin{array}{ll}
S \to q_0 T & \\
T \to \left(\begin{smallmatrix} a \\ a \end{smallmatrix}\right) T & a \in \Sigma \\
T \to U & \\
U \to \left(\begin{smallmatrix} \varepsilon \\ \underline{b} \end{smallmatrix}\right) U & \underline{b} \text{ blank symbol} \\
U \to \varepsilon &
\end{array}$$

$$\begin{array}{ll}
p\left(\begin{smallmatrix} a \\ C \end{smallmatrix}\right) \to \left(\begin{smallmatrix} a \\ D \end{smallmatrix}\right) q & \delta(p, C) = (q, D, 1) \\
\left(\begin{smallmatrix} b \\ E \end{smallmatrix}\right) p\left(\begin{smallmatrix} a \\ C \end{smallmatrix}\right) \to q\left(\begin{smallmatrix} b \\ E \end{smallmatrix}\right)\left(\begin{smallmatrix} a \\ D \end{smallmatrix}\right) & \delta(p, C) = (q, D, -1)
\end{array}$$

$$\begin{array}{ll}
\left(\begin{smallmatrix} a \\ C \end{smallmatrix}\right) q_H \to q_H a\, q_H & \text{halting state} \\
q_H \left(\begin{smallmatrix} a \\ C \end{smallmatrix}\right) \to q_H a\, q_H & \\
q_H \to \varepsilon &
\end{array}$$

Quoi?

One should think of this as a two-track alphabet $\left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right)$ plus some big symbols that take up both tracks.

Block 1: produce a string of the form

$$\begin{array}{cccccccccc}
 & a_1 & a_2 & \dots & a_n & \varepsilon & \varepsilon & \dots & \varepsilon \\
q_0 & a_1 & a_2 & \dots & a_n & \underline{b} & \underline{b} & \dots & \underline{b}
\end{array}$$

Block 2: Run the computation of the Turing machine on the second track; if the padding is long enough, we can accommodate any convergent computation.

Block 3: If the accepting state appears, erase the bottom track and leave the top track (the original input). Done.

□

Erasing versus Monotonicity

Note that it is critical that the grammar can erase all the extra space used during the actual computation, things are wildly non-increasing here.

By contrast, define a grammar to be monotonic if all productions are of the form

$$\pi : \alpha \to \beta \qquad \text{where } \alpha \in \Gamma^\star V \Gamma^\star, \beta \in \Gamma^\star, |\alpha| \le |\beta|$$

As we have seen already, a monotonic grammar can only generate a decidable language.

In fact, these look rather similar to context-sensitive grammars except that we are now allowed to manipulate the context (but see 2 slides down). In particular, every CSG is monotonic.

Monotonic is Context-Sensitive

Theorem

A language is context-sensitive iff it is generated by a monotonic grammar.

Proof.

We have to show that every monotonic language can be accepted by an LBA. Initially, some terminal string $a_1, a_2, \ldots, a_n$ is written on the tape.

- Search handle: the head moves to the right a random number of places, say, to a_i.
- Check righthand side: the machine verifies that $a_i, \ldots, a_j = \beta$ where $\alpha \to \beta$ is a production.
- Replace by lefthand side: the block $a_i, \ldots, a_j$ is replaced by α, possibly leaving some blanks.
- Collect: remove possible blanks by shifting the rest of the tape left.

This loop repeats until the tape is reduced to S and we accept. If any of the guesses goes wrong we reject.

Tiny Example

One can also directly convert monotonic productions to context-sensitive ones. As a simple example, consider a commutativity rule

$$AB \to BA$$

Here are equivalent context-sensitive rules:

$$AB \to AX$$
$$AX \to BX$$
$$BX \to BA$$

Here X is a new variable and the green variable is the one that is being replaced. Note how the left/right context is duly preserved.

Full Rewrite

How about we give up on the distinction between variables and terminals, and use just a single alphabet Σ and productions

$$\pi : \alpha \to \beta \qquad \text{where } \alpha, \beta \in \Sigma^\star$$

So there is no distinction between terminals and variables. Suppose we have a finite list R of such productions. We can still define a notion of derivation:

$$x \to_R y \iff \exists\, u \to v \text{ in } R, r, s\, (x = rus \text{ and } y = rvs).$$

As always, $\overset{*}{\to}_R$ is the reflexive transitive closure $\to_R$.

Definition

The structure $\langle \Sigma, R \rangle$ is a rewrite system or a semi-Thue system or a reduction system.

Axel Thue (1863–1922)

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

Two-Way Rewrite

Rewrite rules are naturally directed: u can be replaced by v, but not necessarily the other way round. Sometimes (in particular in algebra) one wants to think of the rules as equations:

$$u \to v \qquad \text{versus} \qquad u = v$$

Define the symmetric closure of R to be

$$R^s = R \cup R^{\text{op}}$$

We can think of R^s as a system of equations between words.

Analogously, one defines $x \leftrightarrow_R y$ and $x \overset{*}{\leftrightarrow}_R y$ by using the rules in R^s rather than just R. In this case one speaks of a Thue system.

Losing Direction

If you prefer, you can think of this as a compact description of (infinite) graphs over the vertex set $\Sigma^\star$.

The step from R to R^s is then very similar to turning a digraph into a ugraph by ignoring the direction of the edges.

We can think of R as defining a digraph on $\Sigma^\star$: the edges are given by $x \to_R y$.

Using R^s instead will produce a ugraph and the equivalence relation $x \overset{*}{\leftrightarrow}_R y$ corresponds to the connected components of this graph.

Example: Erasing Things

Consider alphabet $\{a, b\}$ and R: $ab \to \varepsilon$, $ba \to \varepsilon$.

Recall our old counting function $f(x) = |x|_a - |x|_b$.

Claim

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

Proof.

Every application of $\to_R$ 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, $x \to_R uv$, done by induction.

□

Mystery System

$$P\colon \quad abc \to ab, bbc \to cb$$

Here are the first few steps applying P^s to abb.

abb

$abcb$

$abb, abbbc, abccb$

$abcb, abcbbc, abcccb$

$abb, abbbc, abccb, abbbcbc, abccbbc, abccccb$

$abcb, abcbbc, abcccb, abbbbbcc, abcbbcbc, abccbbc, abcccccb$

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

Come Again?

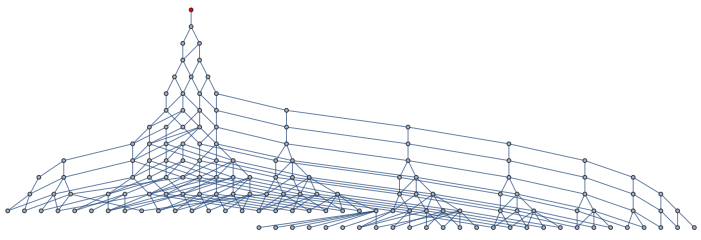
This seems fairly chaotic, but if one focuses on strings of the form $a^\star b^\star c^\star$ there is a pattern. Write L for all the strings derivable from abb. Then

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

This shows that L is not context-free (but easily context-sensitive).

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

150



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

Undecidability

Since a Chomsky grammar is automatically a rewrite system, we inherit all the undecidability results from there.

More directly, since configurations of a Turing machine can be written as strings of the form

$$C = b_m b_{m-1} \ldots b_1\, p\, a_1 a_2 \ldots a_n$$

the one-step relation can be described as a rewrite system R, so $\overset{*}{\to}_R$ corresponds to computations of the TM.

Thus, we cannot decide whether $x \overset{*}{\to}_R y$, only semidecide.

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

Rewriting and Computer Algebra

Mathematica has a large collection of built-in rewrite rules that can simplify mathematical expressions:

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

A (rather lame) trace of the corresponding evaluation:

$$(((3x-\pi,-\pi+3x),\cos(-\pi+3x),-\cos(3x)),2(-\cos(3x)),2(-1)\cos(3x),$$
$$-2\cos(3x),-2\cos(3x)),\frac{\partial(-2\cos(3x))}{\partial x},6\sin(3x),-(6\sin(3x)),$$
$$-6\sin(3x),-6\sin(3x),6\sin(3x)-6\sin(3x),-6\sin(3x)+6\sin(3x),0$$

Rewriting and Functional Languages

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]
```

Requirements

To use rewrite rules as simplifiers in the real world one typically needs a few special properties:

- The precise order in which the rules are applied should not matter. Otherwise it gets tricky to determine the right rewrite order.
- Whatever rules we apply, after finitely many steps we should come to a fixed point. Otherwise we don't get termination.

So the ultimate result of rewriting should be uniquely determined by the input; any term must reduce to a unique expression, independent of any choices along the way.

Irreducibility and Normal Forms

Definition

$x \in \Sigma^\star$ is irreducible if there is no y such that $x \to_R y$.

A normal form for x is an irreducible word x' such that $x \overset{*}{\to}_R x'$.

Normal forms are particularly useful if they are unique: we can use x' as "the name" for the class of x with respect to $\overset{*}{\leftrightarrow}_R$.

In the example above, x such that $f(x) \geq 0$ has normal form $a^{f(x)}$, and $b^{-f(x)}$, otherwise.

Hence the equivalence classes under $\overset{*}{\leftrightarrow}_R$ behave much like the integers. We have a weird description of $\mathbb{Z}$.

Irreducible is Regular

Lemma

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

Proof.

Irreducibility only depends on the left hand sides of rules $(\ell, v) \in R$.

Suppose ℓ_1, ℓ_2, ..., ℓ_n is a listing of all LHS.

We can remove all ℓ_i that have some other ℓ_j as proper factor.

Then a word x is irreducible iff $x \notin \Sigma^\star\{\ell_1, \ldots, \ell_n\}\Sigma^\star$ and it is not hard to construct a FSM for this (even a deterministic one, using, say, the Aho-Korasick algorithm).

□

Another Example

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

$$R: \quad aaaa \to \varepsilon, bb \to \varepsilon, ba \to aaab$$

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.

Another Example

Otherwise apply the last rule to get

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

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

Done by induction.

We are really dealing with the dihedral group D_4: a corresponds to a rotation by $\pi/2$ and b corresponds to a reflection (horizontal, vertical or diagonal, it does not matter).

Confluence

Rewrite systems are almost always nondeterministic: there are many different ways a string can be rewritten. In order for a unique normal form to exist one would like to be in a situation where the order of rule applications does not matter.

Definition

A rewrite system is confluent if for any $w, x, y \in \Sigma^\star$ such that $w \xrightarrow{*} x$ and $w \xrightarrow{*} y$ then there is a $z \in \Sigma^\star$ such that $x \xrightarrow{*} z$ and $y \xrightarrow{*} z$.

The system is Church-Rosser if for any $x, y \in \Sigma^\star$ such that $x \overset{*}{\leftrightarrow} y$ there is a $z \in \Sigma^\star$ such that $x \xrightarrow{*} z$ and $y \xrightarrow{*} z$.

Lemma

A rewrite system is Church-Rosser iff the system is confluent.

Termination

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 if there is no infinite sequence $(x_i)_{i<\omega}$ of words such that $x_i \to_R x_{i+1}$.

Theorem

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

Proof.

Rewrite w until no further reductions are possible (by termination), call the result w', a irreducible string.

Proof Cont'd.

Now suppose there is some other normal form w''. If $w' \neq w''$ then at least one of them can be rewritten by confluence, contradicting irreducibility.

□

In fact, each rewrite step can be handled by a finite state machine: by cleaning up LHS as in the lemma, for each reducible w there is a unique "handle" ℓ such that $w = x\ell y$, $|x|$ minimal, which can be found by a FSM and replaced by v such that $\ell \to v$ is in R. This can easily be handled by a transducer.

But note that there is no general bound on the number of rewrite steps.

Insuring Termination

A standard way to guarantee termination is to construct some well-ordering $>$, a so-called reduction order, such that

$$(u, v) \in R \quad \text{implies} \quad u > v$$

Length-lex order is often helpful: $x <_{\text{ll}} y$ if $|x| < |y|$ or $|x| = |y|$ and $x < y$ in lexicographic order.

Note that length-lex order is a well-ordering by brute force. It is often more useful in induction arguments than lexicographic order.

Termination versus Algebra

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

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

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.

- General Grammars
- Rewrite Systems
- 3 Thue Systems
- Post Systems

The Word Problem

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.

The Word Problem

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.

Words and Equations

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 $\langle \mathbb{N}^2, +, 0 \rangle$.

Free Monoids and Quotients

Lemma

The Thue equivalence $\overset{*}{\leftrightarrow}_R$ is a congruence on $\Sigma^\star$.

Proof.

It is clear that $\overset{*}{\leftrightarrow}_R$ is in fact an equivalence relation.

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

□

But then we can form the quotient monoid

$$M_R = \Sigma^\star / \overset{*}{\leftrightarrow}_R$$

Multiplying Quotients

More precisely, since $\overset{*}{\leftrightarrow}_R$ 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 $M_R = \Sigma^\star / \overset{*}{\leftrightarrow}_R$.

Finite Representations

In this context Σ is called a set of generators for M_R.

It is easy to see that every monoid M can be written in the form M_R (just use M as Σ and let R be all valid equations in M).

Of course, this is not particularly exciting. The interesting case is when Σ is finite and R is finite, in which case we speak of a finite representation.

Given a finite representation we can easily check $x \leftrightarrow_R y$ by table lookup.

But how about the word problem in M_R, which comes down to checking $x \overset{*}{\leftrightarrow}_R y$?

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

Undecidability

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:

$$R: \quad ccbb \to bbcc, bcccbb \to cbbbcc, accbb \to bba,$$
$$abcccbb \to cbba, bbccbbbbcc \to bbccbbbbcca$$

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.

Post Tag System

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

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

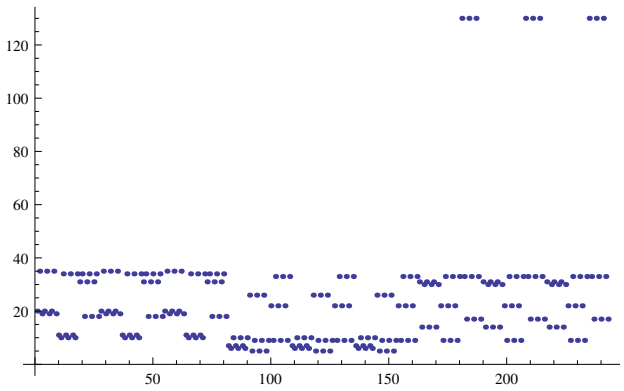
For example, we have the following, rather circuitous, derivation from aaa to a:

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

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

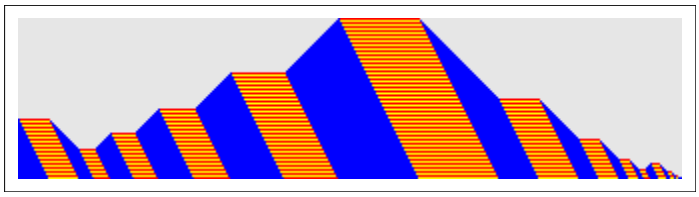
In this system, all initial words produce a terminating derivation.

Orbit Lengths



The length of all orbits to the fixed point, for all 243 words of length 5.

The Orbit of a^{30}



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 ...

Basic Behavior

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.

Undecidability

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.

Undecidability in the Real World

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

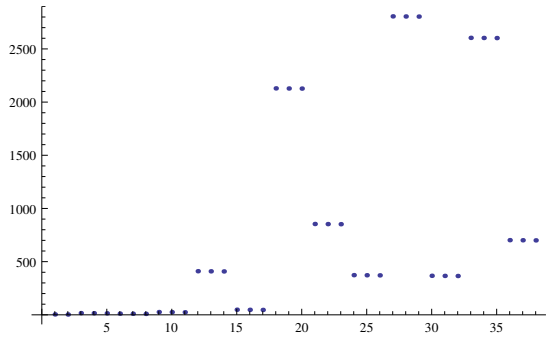
But for the $d = 3$ he was stumped by the following binary system:

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

This may seem too simpleminded to cause any serious problems: for cryinoutloud, there are just two tiny rules; with a bit of effort we should be able to completely understand this system.

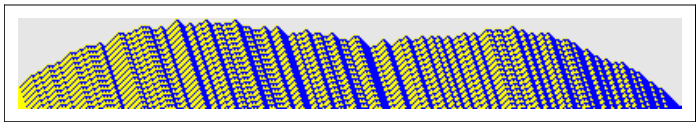
Alas ...

Transient Lengths



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

The Orbit of b^{14}



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

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