CDM

Automatic Sequences

Klaus Sutner
Carnegie Mellon University

70-auto-seq 2017/12/15 23:20

Outline

Generating Infinite Words

We can think of an infinite word $A \in \Gamma^\omega$ as a function $A : \mathbb{N} \to \Gamma$.

Words where this function is computable are particularly interesting.

Pushing things one step further: how about words where the function can be computed by a finite state machine?

Note that we are interested in individual words here, not the recognition of sets of words as in the setting of Büchi automata. We will not scan the word but instead study machines that on input n output $A(n)$. Of course, we need to explain exactly how these machines work.

Base k Representations

The input alphabet will be a digit alphabet of the form

$$\Sigma_k = \{0, 1, \ldots, k-1\}$$

Recall the standard base k (or radix k) representation of a natural number: the leading digit is the MSD:

$$\text{rep}_k(n) = d_r d_{r-1} \ldots d_0 \text{ where } n = \sum_{i \le r} d_i B^i, d_r \neq 0$$

For the sake of completeness let us fix the empty word as the representation for zero (though the letter 0 also makes sense).

Similarly, given an arbitrary word w over Σ_k, we will denote its value in base k by $\text{val}(w)$. Note that $\text{val}(0w) = \text{val}(w)$, so this notation system is not unique.

Reverse Base k Representations

Alternatively, we can consider the leading digit to be the LSD, leading to reverse base k (or reverse radix k) notation:

$$\text{rrep}_k(n) = d_r d_{r-1} \ldots d_0 \text{ where } n = \sum_{i \le r} d_i B^{r-i}, d_r \neq 0$$

Again the empty word represents 0 and we write $\text{rval}(w)$ for the numerical value of a word $w \in \Sigma_k^\star$.

Proposition

$\text{rep}(n)^{\text{op}} = \text{rrep}(n)$

Since regular languages are closed under reversal one may suspect that both notation systems will produce the same results (but see the section on state complexity below).

Partitioned Automata

Since we want to determine $A(n)$ given (the representation of) n we need to consider a slight generalization of ordinary DFAs: machines with multiple final state sets.

Definition

A partitioned deterministic finite automaton (PDFA) is an automaton of the form

$$M = \langle\, Q, \Sigma, \delta; q_0, \boldsymbol{F} \,\rangle$$

where $\langle\, Q, \Sigma, \delta \,\rangle$ is a deterministic and complete transition system, $q_0 \in Q$ an initial state and $\boldsymbol{F} = (F_a \mid a \in \Delta)$ is a partition of Q. The cardinality of Δ is the order of M.

It will be convenient to think of Δ, the coordinates of $\boldsymbol{F}$, as an alphabet.

We are here mostly interested in the case $\Sigma = \Sigma_k$.

Behaviors

Note that a PDFA of order 2 is just a plain DFA. One may expect the machinery of behaviors and quotients to carry over.

For $a \in \Delta$ the a-behavior of a state p is

$$[\![p]\!]_a = \{\, x \in \Sigma^\star \mid \delta(p, x) \in F_a \,\}$$

The behavior of p is the vector $([\![p]\!]_a \mid a \in \Delta)$.

In a slight abuse of terminology, we define the language of M is

$$\mathcal{L}(M) = [\![q_0]\!] \subseteq (\Sigma^\star)^{|\Delta|}$$

The automaton is reduced if its states all have distinct behavior.

Minimal PDFAs

As with ordinary DFAs we are interested in the smallest equivalent (accepting the same language) PDFA.

Theorem

For every PDFA there is an equivalent one that is accessible and reduced. Moreover, this PDFA is unique up to isomorphism.

We will refer to this automaton as the minimal PDFA.

Note that the minimal PDFA can be computed using, say, Hopcroft's algorithm as in the order 2 case.

The Function Angle

On occasion it is more convenient to think of a PDFA as a weak type of transducer that takes values in Δ.

More precisely, we can associate a function $F : \Sigma^\star \to \Delta$ with M via

$$F(x) = a \iff \delta(q_0, x) \in F_a.$$

Note that unlike with general transducers we only evaluate the last state in the computation of M on x; the output of F is a single letter in Δ rather than a word.

Automatic Words

Definition

An infinite word A over alphabet Δ is k-automatic if there exists a PDFA M over alphabets Σ_k and Δ such that for all $w \in \Sigma_k^\star$:

$$\delta(q_0, w) \in F_{A(\mathsf{rval}(w))}$$

We will say that M generates A to distinguish this situation from other forms of acceptance of infinite words.

Thus, our machines operate on reverse base k representations. This is somewhat arbitrary but will be convenient later in our discussion of kernels.

Zeros, LSD and MSD

Note that we allow for trailing zeros; restricting w to $\mathsf{rrep}_k(n)$ would seem to make as much sense.

Also, we might as well consider ordinary base k representations where the first digit is the MSD, with or without leading zeros.

As we will see, none of these choices affect our definition: we always obtain the same class of automatic words.

Robustness

Theorem

Let A be an infinite word. The following are equivalent:

- *A is k-automatic,*
- *A is generated by a PDFA using base k notation,*
- *A is generated by a PDFA using base k notation and admitting leading zeros,*
- *A is generated by a PDFA using reverse base k notation without trailing zeros.*

Leading/trailing zeros are easy to deal with, but the switch to standard base k is more interesting.

Reversal

Since $\mathrm{rrep}_k(n) = \mathrm{rep}_k(n)^{\mathrm{op}}$ we need to deal with reversal.

Suppose M is an accessible PDFA. We will show how to construct the minimal PDFA M^{op} for the (component-wise) reversal of $\mathcal{L}(M)$.

Recall Brzozowski's result that for order 2 this can be done by applying reversal and determinization: M^{op} is isomorphic to $\det(\mathrm{rev}(M))$.

We can generalize this construction to PDFAs as follows.

The General Case

Write

$$M = \langle\, Q, \Sigma, \Delta, \delta, q_0, \boldsymbol{F} \,\rangle$$

for the original PDFAs and set

$$M_a = \langle\, Q, \Sigma, \delta; q_0, F_a \,\rangle \qquad \text{for } x \in \Delta$$
$$M'_a = \det(\mathrm{rev}(M_a))$$
$$M' = \bigoplus_{a \in \Delta} M'_a$$

Attach a partition to M' by setting

$$F'_a = \{\, \boldsymbol{P} \mid q_0 \in P_a \,\}$$

The Minimal PDFA for Standard Base k

Theorem

The PDFA M' is minimal and isomorphic to M^{op}.

Proof. First note that by construction $\mathcal{L}(M') = \mathcal{L}(M)^{\mathrm{op}}$.

A simple induction shows that each state $\boldsymbol{P} = \delta'(\boldsymbol{F}, x)$ of M' forms a partition of Q, so suppose that $\boldsymbol{P} \neq \boldsymbol{R}$, say, $P_a \neq R_a$.

We may safely assume that $p \in P_a - R_a$.

But M is accessible, so $\delta(q_0, x) = p$ for some word x. It follows that x^{op} lies in the a-behavior of $\boldsymbol{P}$ but not of $\boldsymbol{R}$.

Hence M' is both accessible and reduced.

□

Example: Binary Words

As already mentioned, in the special case where

$$\Delta = \mathbf{2} = \{0, 1\}$$

our PDFAs are essentially just ordinary DFAs where $F = F_1$.

The automaton then recognizes the set of positions, written in reverse base k, where the sequence is 1.

Hence every binary regular language that is well-behaved with respect to trailing zeros gives rise to a binary automatic word.

Prouhet-Thue-Morse Word

The PTM word $T = (t_n)$ is a binary word defined by

$$t_0 = 0$$
$$t_{2n} = t_n$$
$$t_{2n+1} = \overline{t_n}$$

Here $\overline{x}$ denotes bitwise complement.

For example, the first 64 bits of T are

0110100110010110100101100110100110010110011010010110100110010110

Iteration

Here is an alternative description of the PTM word. Consider the following sequence of binary words:

$$T_0 = 0$$
$$T_{n+1} = T_n \overline{T_k}$$

Thus T_{n+1} is obtained from T_n by applying the map $\mu(x) = x\overline{x}$.

Since T_n is a proper prefix of T_{n+1} this sequence must have a limit $S \in \mathbf{2}^\omega$. Moreover, S must be a fixed point under μ. But then S satisfies the same recurrence equations as T, so $S = T$.

Proposition

The PTM word T is the limit of the T_n.

Digit Sums

Here is yet another description of the PTM word, one that shows that the word is indeed 2-automatic.

Recall the *binary digit sum* of a natural number n:

$$\mathsf{ds}(n) = \text{ number of 1's in the binary expansion of } n$$

Proposition

$t_n = \mathsf{ds}(n) \bmod 2$

Proof. $\mathsf{ds}(2n) = \mathsf{ds}(n)$ and $\mathsf{ds}(2n+1) = \mathsf{ds}(n) + 1$ □

Decimation

Definition

Let $A : \mathbb{N} \to \Gamma$ be an infinite word. Given a *stride* $s \geq 1$ and on *offset* $d \geq 0$ we can define the decimation of A with respect to s and d by

$$A[s,d](i) = A(s \cdot i + d)$$

Example

For the PTM word T we have $T[2,0] = T$ but $T' = T[2,1] \neq T$. Moreover, $T'[2,0] = T'$ and $T'[2,1] = T$.

A Right Action

We can think of decimation as a right action operating on Γ^ω.

Proposition

The operation $*$ *defined by*

$$[s,d] * [s',d'] = [ss', sd' + d]$$

induces a monoid structure on $\mathbb{N}^+ \times \mathbb{N}$ *with neutral element* $[1,0]$. *We refer to this monoid as the decimation monoid.*

The algebraically inclined will recognize this monoid as a semidirect product of the multiplicative monoid on $\mathbb{N}^+$ and the additive monoid on $\mathbb{N}$.

Proposition

The decimation monoid naturally acts on the right on Γ^ω.

Recovering Letters

Let $0 \leq n < k^i$. Then

$$A(n) = \mathsf{fst}(A[k^i, n])$$

Thus, we can recover the letters of the word from the first letters of the various decimations.

This is particularly interesting when the total number of these decimations is finite.

The k-Kernel

Definition

The k-kernel of a word $A \in \Gamma^\omega$ is defined by

$$\mathsf{Ker}_k(U) = \{\, A[k^i, j] \mid 0 \leq i, 0 \leq j < k^i \,\}$$

As we have seen, the 2-kernel of the PMT word T consists of T and $T[2,1]$.

Thus, the 2-automatic word T has a finite 2-kernel (don't get distracted by the fact that kernel actually has cardinality 2). Could this be coincidence?

Finite Kernels

Theorem

A sequence is k-automatic if, and only if, its k-kernel is finite.

Before we give a proof, consider a PDFA M generating a k-automatic sequence A.

For the sake of this argument, let us interpret the behavior $[\![p]\!]$ of a state p in M to be the word generated with p as the initial state. Then we have

$$[\![\delta(p,d)]\!] = [\![p]\!][k,d]$$

To see this, recall that M is working on LSD-first representations of numbers.

Word Action

Correspondingly we can define a right action of $\Sigma_k^\star$ on Γ^ω by

$$U[w] = U[k^{|w|}, \mathsf{rval}(w)]$$

Thus

$$[\![\delta(q_0,w)]\!] = A[w]$$

by induction on w.

Thus the states in M correspond to the decimations of A with strides k^i.

Proof (of theorem)

First assume A is k-automatic via a PDFA M on m states. Choose a transfer sequence x_p for every state p in M. As we have seen, every kernel element B is of the form $B = A[w]$ for some word w. But then $B = A[x_{\delta(q_0,w)}]$ and the kernel has cardinality at most m.

For the opposite direction consider the finite kernel K of A. Define a PDFA M by

$$\langle\, K, \Sigma_k, \delta, A, \boldsymbol{F} \,\rangle$$

where $\delta(B,a) = B[a]$ and $F_a = \{\, B \mid \mathrm{fst}(B) = a \,\}$. □

Uniform Morphisms

Since kernels naturally correspond to Moore automata generating automatic words using reverse base k representations, the question arises whether there is some natural combinatorial characterization of automaticity that is based on standard base k representations. Voila.

A morphism $\mu : \Gamma^\star \to \Gamma^\star$ is k-uniform if $|\mu(a)| = k$ for all $a \in \Gamma$. μ is extensible if there is a letter $a \in \Gamma$ such that $\mu(a) = au$.

Note that every extensible morphism must have a fixed point:

$$A = a\,u\,\mu(u)\,\mu^2(u)\,\mu^3(u)\ldots$$

If the morphism is also k-uniform then a_n is mapped to the block $a_{kn}a_{kn+1}\ldots a_{kn+k-1} = \mu(a_n)$ under μ.

Morphisms and Automaticity

Theorem

A sequence is k-regular if, and only if, it is the image of a fixed point of a k-uniform morphism under an alphabetic substitution.

Proof.

Let $\mu : \Gamma \to \Gamma^k$ be a k-uniform morphism with fixed point A and $\sigma : \Gamma \to \Sigma_k$ a substitution.

We have to show that $B = \tau(A)$ is k-automatic. Define a PDFA by using Γ as state set:

$$M = \langle\, \Gamma, \Sigma_k, \delta, a, \boldsymbol{F} \,\rangle$$

where $\delta(p,i) = \mu(p)_i$ (assuming 0-indexing) and $F_a = \sigma^{-1}(a)$.

An easy induction shows that $\delta(a, \mathsf{rep}_k(n)) = A(n)$.

Proof, contd.

For the opposite direction assume we have a PDFA

$$M = \langle Q, \Sigma_k, \delta, q_0, \boldsymbol{F} \rangle$$

that generates a word $A \in \Delta^\omega$. We may safely assume that $\delta(q_0, 0) = q_0$.

We use Q as alphabet and define the morphism $\mu : Q \to Q^k$ by

$$\mu(p) = \delta(p,0)\delta(p,1)\dots\delta(p,k-1) \in Q^k$$

Define the substitution τ by $\tau(p) = a \iff p \in F_a$.

Then $A = \tau(\lim \mu^i(q_0))$.

State Complexity

We can use the size of the minimal PDFA generating a k-automatic word as a measure of its complexity.

Definition

For any k-automatic sequence A we refer to the number of states of the minimal PDFA generating A as the state complexity of A.

Proposition

The state complexity of a k-automatic word is the size of its k-kernel.

Exponential Gap

Recall that we use reverse base k as the default representation.

As we have seen, there is also a minimal PDFA M^{op} that generates A using standard base k. We refer to the number of states of M^{op} as the state co-complexity of A.

As we have seen, complexity and co-complexity are exponentially related.

Alas, there may indeed be an exponential gap between the two measures:

Example

Consider the binary word A_r defined by the language

$$0^\star 1 \mathbf{2}^\star 1 \mathbf{2}^r 10^\star$$

Then the state complexity of A_r is $2^{r+2} + 1$, but the co-complexity is $r + 3$.

Computing State Complexity

Suppose we are given a k-automatic word A. How can we compute its state complexity and the corresponding minimal PDFA?

Of course, this depends greatly on the representation of A.

For the time being, let us suppose we can directly manipulate infinite words. In particular we need to be able to compute decimations $A[k^i, j]$ and check them for equality.

Then we can compute the kernel of A and actually the minimal PDFA using the standard closure algorithm: close $\{A\}$ under the operations $X \mapsto X[k, j]$, $0 \le j < k$.

The Kernel Algorithm


```
// generate kernel

K = {A};
set A active;

while( some active B remains )
       deactivate B;
       foreach j < k do
            X = B[k,j];
            if( X notin K )
                add X to K;
                set X active;
            else
                transition (B,j,Y);
```

Initial Segments

Again, there are only two non-logical operations in the algorithm:

- decimation (to compute `X` from `B`),
- equality testing (to check if `X` is in `K`)

But what if we start with a finite prefix $\alpha \sqsubset A$ rather than A itself?

Decimation extends to an operation over finite words in the obvious way. However, the length of the prefix shrinks by a factor of k at each step

To test equality we compare prefixes: $x =_p y$ if $x \sqsubseteq y$ or $y \sqsubseteq x$. Thus, the shorter word has to be a prefix of the longer.

Note that $\alpha, \beta \sqsubset A$ implies $\alpha =_p \beta$.

Approximations

For $\alpha \sqsubset A$ we have $\alpha[w] \sqsubset A[w]$, but it may well happen that $\alpha[w] =_p \alpha[v]$ when in fact $A[w] \neq A[v]$.

Running the kernel algorithm with these modified non-logical operations will thus in general produce an under-approximation, we may obtain false identities.

The question then is: how long a prefix of A is required to produce the actual kernel?

More precisely, how long does the prefix have to be in order to

- determine the size of the kernel,
- determine the minimal PDFA.

Approximation Theorem

Theorem

Let A be k-automatic with state complexity m. The prefix of length k^{2m-3} of A suffices to determine the state complexity, and the prefix of length k^{2m-2} suffices to determine the minimal PDFA. Moreover, both bounds are tight.

Proof. Suppose the k-kernel of A is $K = \{A = A_1, \ldots, A_m\}$.

There are witnesses $v_i \in \Sigma_k^\star$ of length at most $m-1$ such that $A_i = A[v_i]$.

Moreover, for $i \neq j$ there are discriminating words $w_{i,j}$ of length at most $m-2$ such that $\mathsf{fst}(A_i[w_{i,j}]) \neq \mathsf{fst}(A_j[w_{i,j}])$.

It follows that the kernel algorithm from above will produce correct number of kernel elements given a prefix of length k^{2m-3}. To obtain the whole PDFA we need the first k^{2m-2} letters.

Proof, contd.

To see that the bounds are tight consider the regular languages

$$L_1 = \{\, x \in \mathbf{2}^\star \mid \#_1 x = r \pmod{r+1} \,\}$$

$$L_2 = \{\, x \in \mathbf{2}^\star \mid \#_1 x = r \pmod{r+2} \,\}$$

$$L_3 = \{\, x \in \mathbf{2}^\star \mid \#_1 x = r \,\}$$

and write A_1, A_2 and A_3 for the the corresponding binary words.

A_1 has state complexity $r+1$ and A_2 has state complexity $r+2$ but they agree on their first $2^{2m-3}-1$ bits.

Words A_2 and A_3 both have kernels of size $m = r+2$ but distinct PDFAs, yet they agree on their first $2^{2m-2}-1$ bits. □

Summary

- There are automatic sequences.