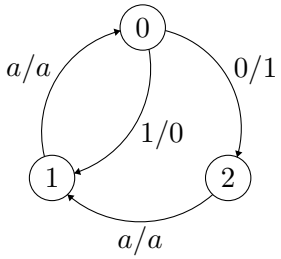
Orbit Checking for Invertible Binary Transducers

Evan Bergeron

April 22, 2016

A Simple Invertible Transducer

This is A_2^3.



Let $\underline{i}$ be the function induced by applying A_2^3 to some string with state i as the start state. $\underline{0}, \underline{1}, \underline{2}$ generate a semigroup under composition. For example, $\underline{0}^2$ is the result of applying $\underline{0}$ twice. We call such functions *transductions*.

If x is some string, call $\underline{0}^* = \{\underline{0}^t(x) \mid t \in \mathbb{N}\}$ the *orbit* of x under $\underline{0}$.

We prove that checking if $y \in \underline{0}^*(x)$ is in P. We then investigate orbit checking for arbitrary f in the semigroup.

A Necassary Condition

Let $f = \underline{0}$. Let $y \in f^*(x)$. Let c be a sequence where c_i is the number of times x_i is flipped on the way to y. Then we claim that

$$\begin{aligned} c_i = \lfloor (c_{i-3}/2) \rfloor + (c_{i-3} \mod 2) \cdot (1 - x_{i-3}) \\ + \lfloor (c_{i-2}/2) \rfloor + (c_{i-2} \mod 2) \cdot x_{i-2} \end{aligned}$$

where c_0 is the index of y in $f^*(x)$, c_1 is 0, and c_2 is $\lfloor c_0/2 \rfloor$.

x_0 is flipped upon every invocation of f. x_1 is never flipped. x_2 is flipped roughly every other time x_0 is flipped. That is, it's flipped every time x_0 changes from a 1 to a 0. Without loss, we may assume $x_0 = 0$. Thus, $c_2 = \lfloor c_0/2 \rfloor$ (it's not flipped the first time, but then is flipped every other time afterward).

Consider some application of f. c_i is flipped iff c_{i-2} is flipped from a 1 to a 0 or c_{i-3} is flipped from a 0 to a 1. If c_{i-2} and c_{i-3} are even, then this is precisely every other flip. If either c_{i-2} or c_{i-3} is odd, then c_i is dependent on both c_{i-2} and c_{i-3} as well as the initial conditions in x. We add one depending on whether or not the first flip of c_{i-2}, c_{i-3} causes c_i to flip as well.

A Sufficient Condition

Let p be a sequence where $p_i = c_i \mod 2$ for all i. Then if $x \oplus y$ looks like some p, then $y \in f^*(x)$. That is, if you fix your initial conditions and the above recurrence holds through $x \oplus y$, then $y \in f^*(x)$.

That being said, we necassarily don't know c_0.

$\underline{0}$ Orbit Checking is in NP

Our verifier takes in two strings x, y, and an index i. This index is the position of y in x's orbit. WLOG, suppose $x_0 = 1$. We first set $c_0 = i$, $c_1 = 0$, and $c_2 = \lfloor c_0/2 \rfloor$. We then calculate c_i and check that $c_i \mod 2 = x_i \oplus y_i$ for all i.

The certificate is poly length with respect to x and y, as the orbit of y has length at most 2^n (so the length of an index is at most n).

Semigroups Revisited

Recall that the states of A_2^3 form a semigroup $\mathcal{S}(A_2^3)$. This semigroup is commutative - it doesn't matter what order we apply the transductions in. This follows by commutativity of addition modulo 2 (suffices to consider each bit one at a time). This means that every element in $\mathcal{S}(A_2^3)$ looks like $\underline{0}^i\underline{1}^j\underline{2}^k$ for $i, j, k \in \mathbb{N}$.

Further, we have an identity element in $\mathcal{S}(A_2^3)$:

$$\underline{0}^2\underline{1}^2\underline{2} = I$$

Proof. Proof by induction - it suffices to show that c_i is even for all i. In fact, c_i will be 2.

c_0 is 2, each of the applications of $\underline{0}$ flip x_0 once. The applications of $\underline{1}$ and $\underline{2}$ skip it. c_1 is also 2 as each application of $\underline{1}$ flips x_1 once and $\underline{0}$ and $\underline{2}$ skip it. $c_2 = 2$ - $\underline{2}$ flips it once, $\underline{1}^2$ skips it, and exactly one of the two applications of $\underline{0}$ flip it.

Then $c_{i-2} = c_{i-3} = 2$ by induction, so

$$c_i = \lfloor (c_{i-3}/2) \rfloor + \lfloor (c_{i-2}/2) \rfloor = 1 + 1 = 2$$

□

This means that $\mathcal{S}(A_2^3)$ is already a group ($\underline{0}^{-1} = \underline{0}\underline{1}^2\underline{2}$ and so on). Further, since $\underline{2}$ is expressible in terms of $\underline{0}$ and $\underline{1}$, we have that $\mathcal{S}(A_2^3) = \{\underline{0}^i\underline{1}^j \mid i, j \in \mathbb{Z}\}$, giving us a concise data structure to represent elements of $\mathcal{S}(A_2^3)$.

Residuals

If f_u is the transduction corresponding to some state state u of a transducer, then $\partial_b(f_u) = f_v$ if state u transitions to state v on input b. Each transducer has a corresponding derivative table, which is effectively an adjacency list representation of the transducer.

For example, A_2^3 has the following representation:

- $\partial_0(\underline{0}) = \underline{2}$
- $\partial_1(\underline{0}) = \underline{1}$
- $\partial_b(\underline{2}) = \underline{1}$ for any b
- $\partial_b(\underline{1}) = \underline{0}$ for any b
- $\partial_0(\underline{0}^i) = \underline{1}^{\lfloor i/2 \rfloor}\underline{2}^{\lceil i/2 \rceil}$
- $\partial_1(\underline{0}^i) = \underline{1}^{\lceil i/2 \rceil}\underline{2}^{\lfloor i/2 \rfloor}$
- $\partial_b(\underline{1}^i) = \underline{0}^i$ for any b
- $\partial_b(\underline{2}^i) = \underline{1}^i$ for any b
- $\partial_0(\underline{0}^i\underline{1}^j\underline{2}^k) = \underline{0}^j\underline{1}^{\lfloor i/2 \rfloor + k}\underline{2}^{\lceil i/2 \rceil}$
- $\partial_1(\underline{0}^i\underline{1}^j\underline{2}^k) = \underline{0}^j\underline{1}^{\lceil i/2 \rceil + k}\underline{2}^{\lfloor i/2 \rfloor}$

We can extend this notion of differentiation to differentiating with respect to arbitrary strings, rather than single bits (simply iterate the differentiation). Then, for $x, y \in \mathbf{1}^*$, $f \in \mathcal{S}$

$$f(xy) = f(x)\partial_x f(y)$$

A_2^3 Orbit Checking is in NP

Call $f \in \mathcal{S}(A_2^3)$ *even* if f leaves the first bit of its input unchanged and *odd* otherwise.

It then suffices, given i as the index into the orbit as a certificate, $f \in \mathcal{S}(A_2^3)$ and strings x, y, to simply iterate through x, differentiating f as we go and asserting that $y_i = \partial_z f(x_i)$, where z is the prefix of x so far.

Parity checking f can be done in polynomial time - simply check the parity of $\underline{0}$ in f. Additionally, differentiation can be performed in polynomial time, as it's a couple of additions and a couple bit shifts. Since the value of i is at most 2^n where n is the input length, the length of i is polynomial with respect to x and y.

The A_2^3 Orbit Relation is Rational

Residuation as Moving Pebbles

We can think of being given a set of pebbles that sit on states in our automaton. Then the residuation operation simply moves these pebbles from state to state. This gives us an easy proof that for fixed f and t, computing f^t is rational. (Now, of course, this follows by closure of rational languages, but it's nice to have a natural, constructive proof).

Generating Pebbles

If we're not given t ahead of time, it's our responsibility to deduce it. We introduce a function π. If $f = (f_0, f_1)$, where f_0 and f_1 are the residuals of f with respect to 0 and 1, respectively and we're sitting on some bit a, then

$$\pi(f) = \begin{cases} f_a f_{\neg a} & \text{if } f \text{ is odd} \\ f_a & \text{if } f \text{ is even} \end{cases}$$

This is part of the quotient operation. Consider moving across two strings simultaenously. After eating the first character, we only need to consider if the suffix of one is in the orbit of the suffix of the other (up to some shift). But our orbit-generating function has changed. The π function tracks how this function changes over time.

The orbit of π over some string corresponds closely with the set of quotients hit in an automaton. (Really, the quotients are a tuple of π's, along with some shift

factor). But then we have that the orbit of π is finite iff the quotients are finite. So we have a necassary and sufficient condition for orbit rationality.

If $\pi^*(f, s)$ is the orbit of π over s with initial condition f, then some automaton A's orbit relation is rational iff $\bigcup_{s \in \Sigma^*} \pi(f, s)$ if finite for all $f \in \mathcal{S}(A)$.

So the question becomes: for which automata is this true?

Note that in the odd case, this corresponds with adding a pebble on the automaton. In the even case, no extra pebbles are added. So it's sort of asking, if given some initial configuration, unioning over all input string pairs in the language, we will only look at finitely many pebble configurations (up to rewrite rules).

In the case where the group is Abelian, function composition is just adding the pebble counts together.

Partial Quotient Trees

Fix some CCC automaton A and some $f \in \mathcal{S}(A)$. We have that orbit checking f is rational iff the number of partial quotients is finite for all $s \in \mathbf{2}^*$. We define a rooted tree that represents all partial quotients reachable from f under from some s.

For each node $f' = (f_0, f_1)$ in the tree, if f' is even, f' has two children, f_0 and f_1. If f' is odd, f' has a single child $f_0 f_1$.

Orbit Equivalence

In addition to the normal group identity equivalence classes, we have cases where two functions f and g are orbit-equivalent. This happens precisely when g is some odd multiple of f (as then g is f^t for odd t, which is coprime to the length of the orbit, and so generates the entire orbit).

For each of these equivalence classes, we have the natural representative being the f of least weight (least number of pebbles).

Lemma 1. *Orbit checking for f is rational iff f's partial quotient tree is regular.*

1-Tree Transducers

We can consider a more general class of transducers than CCC's. Let S be the set of DAGs with a single vertex of out-degree 2 and the rest of out-degree one. We

define the 1-toggle-1-split machines to the corresponding set of transducers. The vertex of out-degree 2 is the toggle state. Every other vertex is a copy state. Let v be the vertex of out-degree 2.

If both of v's out-going edges are in directed cycles containing v, of lengths n and m, respectively, $\mathcal{S}(G) \cong \mathcal{S}(A_m^n)$. The transduction induced by some vertex $u \in G$ is equivalent to $\underline{dist(u,v)} \in \mathcal{K}_m^n = A_m^n$. So given a multiset of vertices in G, we easily have a word in $\mathcal{S}(A_m^n)$.

If neither of v's out-going edges are in directed cycles containing v, then $\underline{v}$ simply toggles the first bit and for $u \in G$, $\underline{u} \in \mathcal{S}(G)$ simply toggles the $dist(u,v)$-th bit. So given a multiset M of vertices in G, simply take the vertices repeated an odd number of times. Then for $u \in M$, toggle the $dist(u,v)$-th bit. This orbit has length 2.

If exactly one of v's out-going edges are in directed cycles containing v, then let the directed cycle have length n. Call this class of machines the **1-tree transducers**.

Then given $u \in G$, $\underline{u}$ performs either addition or subtraction on the natural number $\{x_{d+kn} \mid d = dist(u,v), k \in \mathbb{N}\}$ written in LSB-first form, mod the size of the set. In this case, given a multiset $M = \{u_1^{k_1} \dots u_i^{k_i}\}$ of vertices in G, we may obtain a action multiset $\{d_1^{k_{i_1}+\dots+k_{i_{j_1}}}, \dots d_k^{k_{i_k}+\dots+k_{i_{j_k}}}\}$. (This looks complicated, but simply take $dist(u,v)$ for each $u \in M$ and add the powers together).

Consider all the substrings of x mod i. Define $x(i) = \{x_{i+kn}\}_{k\in\mathbb{N}}$. Then

$$x(i) = y(i) \pm \sum_{d_j \in [i]_n} 2^{\lfloor d_j/n \rfloor} \cdot k_j$$

COROLLARY 1. *From the above, we easily get that 1TT semigroups are Abelian.*

CONJECTURE 1. *In general, 1TT semigroups are not groups - there's no clear inverse. Sure, for a fixed length string, we work modulo some power of two. But the inputs can be arbitrarily long.*

CONJECTURE 2. *If A is a 1TT with a cycle of length n, then $\mathcal{S}(A) \cong \langle \mathbb{N}^n, + \rangle$, where addition is component-wise?*

PROPOSITION 1. *1TT orbit checking can be done in polynomial time.*

Proof. Given a word in $\mathcal{S}(G)$ and two strings x, y, determine $x(i) - y(i)$ for $i \in [n-1]$. Then simply verify that each of these differences is the same multiple of $S_i := \sum_{d_j \in [i]_n} 2^{\lfloor d_j/n \rfloor} \cdot k_j$.

We may assume the word we're given is given as a list of tuples (d, k) where d is the distance from the split vertex and k is the number of pebbles on that vertex. The number of bits of any arithmetic expression is bounded by the lengths of x, y, as we're working mod $2^{|x|}$. So all of these arithmetic computations are polynomial.

TODO - are there polynomially many S_i's? Equivalently, is the cycle length polynomial wrt the input? Probably, but worth noting. □

LEMMA 2. *Given a word w in $\mathcal{S}(A)$, where A is some 1-tree transducer, the orbit length of w is the lcm of all the S_i's.*

THEOREM 1. *For a fixed 1TT A and a fixed $f \in \mathcal{S}(A)$, orbit checking is rational.*

Proof. The plus-one relation is rational, so any plus-k relation is also rational. So for a fixed $f \in \mathcal{S}(A)$, we can determine the S_i's. Then we can make plus-S_i transducers. Then we can "interleave" the transducers to switch between which $x(i)$ we want to consider. □

COORDINATE SYSTEMS

$\forall x \in 2^{2k} \exists 0 \leq a, b \leq 2^k (x = 0^a 1^b (0^{2k}))$.