CDM Homework Examples

0.1. Magic Words (XX)

Background The great magician Chidur poses the following problem to his apprentice. Being exceedingly powerful, Chidur can associate words with special properties. In this case, he has chosen to associate one of three properties with each word: healing, neutral or deadly. The apprentice is fairly new, so Chidur only uses words over the uppercase Latin alphabet $\{A, B, \ldots, Z\}$, words like "ABRACADABRA". Chidur abhors the empty word. The notation x^{op} stands for the word x written backwards.

Alas, Chidur refuses to explain directly how he has distributed the properties. All he will tell the hapless apprentice is this:

- Every word has exactly one property.
- Some pairs of words are exalted. In particular:
  - All pairs (XwX, w) are exalted where w is any word.
  - If (u, v) is exalted then (Yu, Xv) is also exalted.
  - If (u, v) is exalted then (Zu, v^{op}) is also exalted.
  - If (u, v) is exalted then (Wu, vv) is also exalted.
- If (u, v) is exalted and u is neutral then v is deadly.
- If (u, v) is exalted and u is deadly then v is neutral.

Task

A. Describe a formal model for this problem and explain how your model correctly represents healing words.

B. Find all shortest healing words.

0.2. Dual Formulae (XX)

Background

For this problem let us only consider Boolean connectives $\neg$, $\vee$ and $\wedge$ (no implication, xor etc.) For any propositional formula φ dual φ^{op} to be obtained from φ by interchanging $\wedge$ and $\vee$, as well as $\bot$ and $\top$. For example, $(p \vee \neg q)^{\mathsf{op}} = p \wedge \neg q$.

Similarly, for a truth assignment σ, define $[\![p]\!]_{\sigma^{\mathsf{op}}} = [\![\neg p]\!]_{\sigma}$ (i.e., flip the truth values for all propositional variables), and then generalize inductively to all formulae.

A formula φ is self-dual if φ and φ^{op} are equivalent. For example, p and $\neg p$ are both self-dual but $p \wedge q$ is not.

Task

A. What would the dual of a biconditional $\varphi \Leftrightarrow \psi$ be?

B. Show that the dual is an involution: $(\varphi^{\mathsf{op}})^{\mathsf{op}} = \varphi$.

C. Show that $[\![\varphi^{\mathsf{op}}]\!]_{\sigma} = [\![\neg\varphi]\!]_{\sigma^{\mathsf{op}}}$.

D. Show that φ is a tautology if, and only if, φ^{op} is a contradiction.

E. Show that two formulae are equivalent, if, and only if, their duals are equivalent.

F. Find a non-trivial example of a self-dual formula in three variables. Generalize to any odd number of variables.

Comment The first few properties are more or less obvious, but try to come up with clear, elegant proofs. For the last part think about counting. It may help to think about Boolean functions rather than formulae: f is self-dual iff $f(x_1, \ldots, x_n) = \neg f(\neg x_1, \ldots, \neg x_n)$.

0.3. Presburger and Skolem Arithmetic (XX)

Background

Presburger arithmetic is the fragment of arithmetic that has addition but no multiplication; Skolem arithmetic is the fragment that has multiplication but no addition. Both turn out to have decidable first-order theory (in stark contrast to general arithmetic).

It is a folklore result that addition is rational and even synchronous. More precisely, there is a synchronous relation $\alpha(x, y, z)$ that expresses $x + y = z$ where the numbers are given in reverse

binary. There is no corresponding synchronous (or even rational) relation $\mu(x, y, z)$ for multiplication. However, we can choose a different encoding from $\mathbb{N}_+$ to $\mathbf{2}^\star$ that makes multiplication rational. Alas, this encoding is slightly strange and mangles addition badly.

Task

A. Show that the addition predicate α is rational for numbers in reverse binary.

B. Argue that α is actually synchronous.

C. Show that multiplication is not rational for numbers in reverse binary.

D. Concoct an encoding of the positive natural numbers that makes multiplication rational. Hint: think about primes.

E. Exlain why your encoding does not work for addition.

Comment For part (A) you can either write a rational expression or draw a diagram. This is a bit of a pain; whatever you do, try to make it look nice.

For part (C) you may freely use pumping lemmings and the like.

For part (E) no formal proof is necessary, just a reasonable explanation.

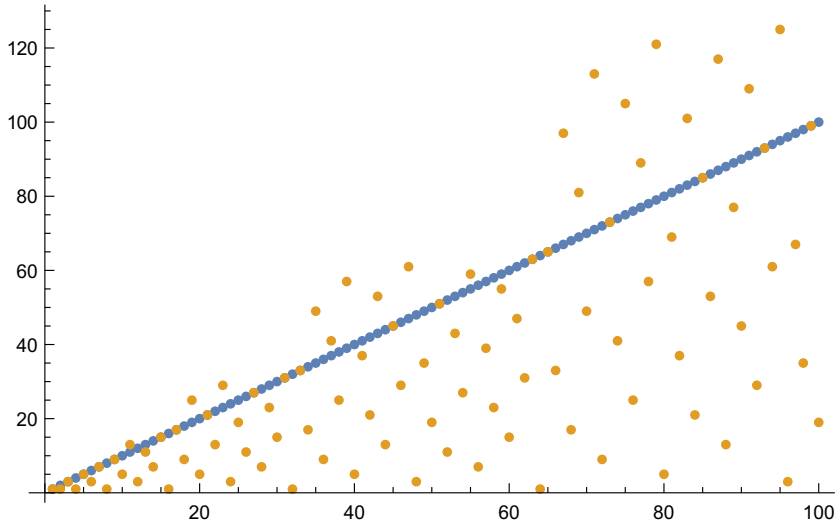
0.4. A Recursion (XX)

Background

Consider the following function f, presumably defined on the positive integers.

$$\begin{aligned} f(1) &= 1 \\ f(3) &= 3 \\ f(2n) &= f(n) \\ f(4n+1) &= 2f(2n+1) - f(n) \\ f(4n+3) &= 3f(2n+1) - 2f(n) \end{aligned}$$

For what it's worth, here is a plot of the first few values.



Task

A. Consider small values of f and conjecture an explicit, non-recursive definition of f.

B. Prove that your definition is correct and conclude that f is indeed a function from $\mathbb{N}_+$ to $\mathbb{N}_+$.

C. Is f primitive recursive?

Comment

Implement f and experiment.

0.5. The Busy Beaver Function (RM) (XX)

Background

The Busy Beaver function β is a famous example of a function that is just barely non-computable. For our purposes, let's define $\beta(n)$ as follows. Consider all register machines P with n instructions and no input (so all registers are initially 0). Executing such a machine will either produce a diverging computation or some output x_P in register R_0. Define $\beta(n)$ to be the maximum of all x_P as P ranges over n-instruction programs that converge.

It is intuitively clear that β is not computable: we have no way of eliminating the non-halting programs from the competition. Alas, it's not so easy to come up with a clean proof. One line of reasoning is somewhat similar to the argument that shows that the Ackermann function is not primitive recursive: one shows that β grows faster than any computable function.

Task

A. Show that, for any natural number m, there is a register machine without input that outputs m and uses only $O(\log m)$ instructions.

B. Assume $f : \mathbb{N} \to \mathbb{N}$ is a strictly increasing computable function. Show that for some sufficiently large x we must have $f(x) < \beta(x)$.

C. Conclude that β is not computable.

D. Prof. Dr. Blasius Wurzelbrunft sells a device called HaltingBlackBox™ that allegedly solves the Halting Problem for register machines. Explain how Wurzelbrunft's gizmo could be used to compute β.

Comment

The bound in part (A) is far from tight in special cases: some numbers m have much shorter programs: think about 2^{2^k}. But, in general $\log m$ is impossible to beat (Kolmogorov-Chaitin program-size complexity). Part (D) says that β is K-computable.

0.6. Placing Problems in the AH (XX)

Background

Let e be an index for a partial computable function $\mathbb{N} \rightharpoonup \mathbb{N}$. We write W_e for the support of $\langle e \rangle$, the set $\{ x \mid \langle e \rangle(x) \downarrow \}$. So $(W_e)_e$ is a listing of all semidecidable sets. By cutting off the computation at σ steps we get a finite approximation $W_{e,\sigma} \subseteq W_e$. In fact, $x \in W_{e,\sigma}$ is primitive recursive. On the other hand, $x \in W_e$ is Σ_1 in the AH, and $x \notin W_e$ is Π_1. Note that

$$x \in W_e \Leftrightarrow \exists\, \sigma\, (x \in W_{e,\sigma}) \qquad \text{and} \qquad x \notin W_e \Leftrightarrow \forall\, \sigma\, (x \notin W_{e,\sigma})$$

so quantifiers are useful to describe levels in the AH: $\exists$ corresponds to unbounded search, and $\forall$ to the negation thereof.

Define

$$\mathsf{FIN} = \{ e \mid W_e \text{ finite} \}$$
$$\mathsf{ALL} = \{ e \mid W_e = \mathbb{N} \}$$

Task

A. Show that every finite set $F \subseteq \mathbb{N}$ is of the form W_e for some e.

B. Find the level of FIN in the arithmetical hierarchy.

C. Find the level of ALL in the arithmetical hierarchy.

D. Show that both sets are undecidable.

Comment

Start with a high-level description, and then rephrase things so that, in the end, there is a string of quantifiers before a primitive recursive condition. The quantifiers tell you where in the AH the problem lies.

No tight lower bounds are needed, but try to find the lowest possible level. For example, claiming that both problems are Σ_{42} is quite useless. The problems actually are complete at their respective levels.

0.7. Semilinear Counting (XX)

Background

It is often stated that "finite state machines cannot count." To a point, this is correct, but there are special cases when a finite state machine is perfectly capable of counting. Here are some fairly involved examples of counting in zero space.

Recall that a set $C \subseteq \mathbb{N}$ is semilinear if it is a finite union of sets of the form $r + m\mathbb{N}$, where $r, m \in \mathbb{N}$ (for $m = 0$ this is just $\{r\}$). Let $U, V \subseteq \Sigma^\star$ be regular languages, $s \in \Sigma$ and $C \subseteq \mathbb{N}$ a semilinear set. Define new languages

$$L(U, C) = \{\, x \mid x = u_1 u_2 \dots u_\ell, u_i \in U, \ell \in C \,\}$$
$$K(U, s, V, C) = \{\, x \mid \#(x = usv, u \in U, v \in V) \in C \,\}$$

Thus, in $L(U, C)$ we collect all words that can be factored into some number $\ell \in C$ terms, each term belonging to U. For $K(U, s, V, C)$, we select words that have some number $\ell \in C$ of (U, s, V) factorizations. For parts (C) and (D) below, let $\Sigma = \{a, b\}$, U the language of all words containing an even number of as, and V the language of all words containing 1 modulo 3 as.

Task

A. Construct the minimal automaton for $L_C = \{\, 0^\ell \mid \ell \in C \,\}$.

B. Show that $L(U, C)$ is regular by constructing a machine.

C. Show that $K(U, s, V, C)$ is regular using closure properties.

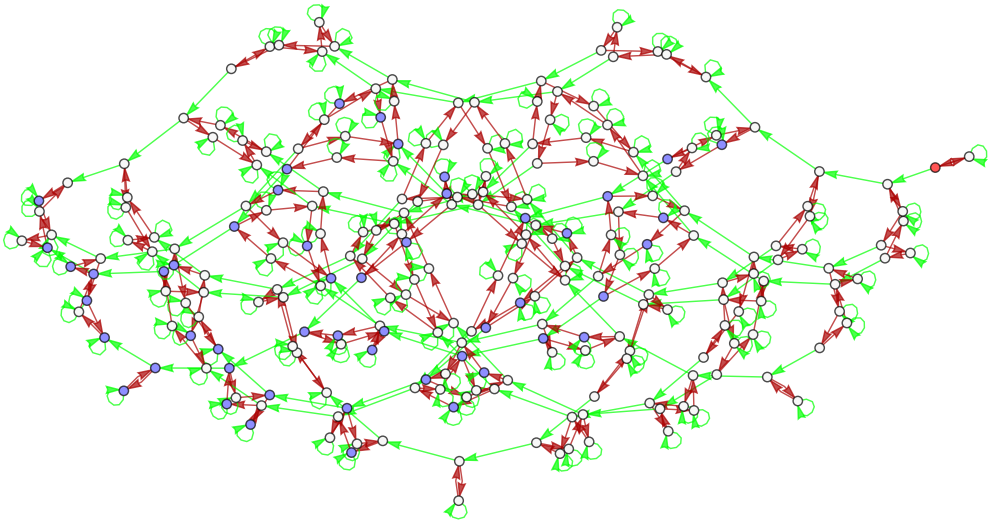
D. Describe the minimal DFA for $K(U, a, V, \{3\})$ in detail.

E. Draw a picture of the natural NFA for $K = K(U, b, V, \{3\})$.

F. Using a pebbling argument on the last NFA, describe a DFA for K.

G. Argue that your DFA for K is already minimal.

Comment For (A) and (B), make sure to take a close look at all the closure properties we talked about; if you pick the right ones, the argument is very short. For part (C), first figure out which words of the form a^ℓ can be factored as required. Needless to say, the last part won't work unless you have built the "canonical" DFA in part (E). For what it's worth, here is a picture of the machine (it has 250 states).



0.8. Palindromes (XX)

Background

A palindrome is a word that reads the same left-to-right and right-to-left. Thus *abba* is a palindrome, but *abab* is not.

Suppose we have a word w over some alphabet Σ. Since single letters are palindromes, we can always find a decomposition

$$w = u_1 u_2 \dots u_k$$

where all the u_i are palindromes. However, it's not so clear how one can minimize the number k of palindromes. This is the palindrome decomposition problem.

Slightly more challenging is the following type of decomposition based on erasing palindromes. Suppose $w = xuy$ where u is a palindrome. Then in one step we can rewrite w to xy. Clearly, after at most $|w|$ steps, we can rewrite w to ε, the empty word. Again, the challenge is to minimize the number of steps. This is the palindrome erasure problem.

Task

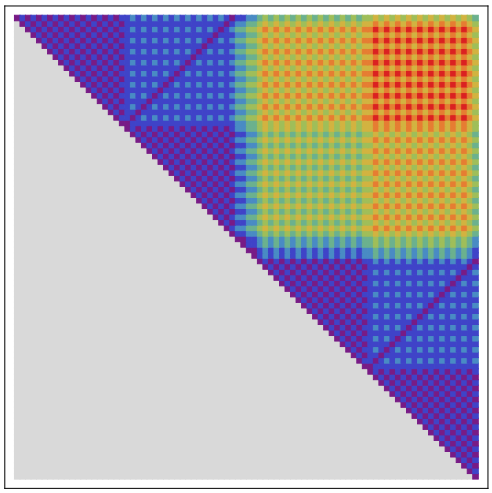
A. Find a cubic time algorithm to solve the palindrome decomposition problem. Your algorithm should return a list of the corresponding palindromes.

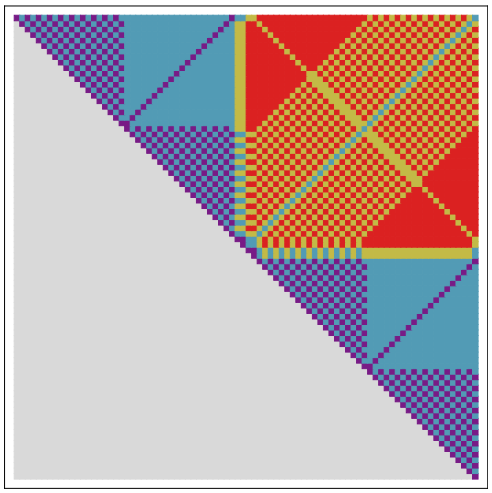
B. Find a cubic time algorithm to solve the palindrome erasure problem. Your algorithm should return a list of the corresponding erasures.

Comment You might find dynamic programming helpful.

For part (B), return a list of pairs (i, j) where $1 \leq i \leq j \leq |w|$ that indicate that one is supposed to remove the palindrome from w_i to w_j. Note that there may be gaps from previous erasures, but keep the original indexing.

Here are pictures for the sorts of tables the come up in the dynamic programming solution.





0.9. MSO and Regular Languages (XX)

Background

According to Büchi's theorem, monadic second-order logic defines exactly the regular languages. However, we gave no proof for the direction "MSO implies regular." Here goes. We are trying to show that for every sentence φ in $\mathsf{MSO}[<]$ the associated language

$$L(\varphi) = \{\, w \in \Sigma^\star \mid w \models \varphi \,\}$$

is in fact regular. The proof uses induction on the buildup of φ. Alas, there is the usual problem with semantics: the components of a sentence typically contain free variables, they are not sentences themselves. To deal with free variables it is best to consider augmented words over the alphabet

$$\Gamma = \Sigma \times (\mathbf{2})^k$$

where k is the number of variables, both first-order and second-order. In other words, we add an appropriate number of binary tracks to the original word that can be used to interpret the free

variables in the quantifier-free part. There is no need for padding, the additional tracks have the same length as w. This is essentially the same machinery as for monadic second-order logic on infinite words discussed in class.

Also recall our formula $\mathsf{even}(X)$ for "X has even cardinality" from class.

Task

A. Prove by induction on the buildup of φ that $L(\varphi)$ is regular. You may safely assume that the formula is in prenex normal form.

B. What can you say about the state-complexity of $L(\varphi)$? Specifically, how large is the machine for the quantifier-free part and what happens when you deal with the quantifiers?

C. Translating $\mathsf{even}(X)$ into a FSM following your inductive definition and the formula from class would produce a large and ugly machine. Construct a small machine directly for this formula.

Comment

For part (B) don't try to get precise results, just a rough estimate of a how large these machines could be compared to the size of the formula. Part (C) is important: sometimes one can streamline machines quite a bit compared to the cookie-cutter result from the algorithm. In particular it may be helpful to enlarge the language by certain predicates with pre-defined machines.

0.10. Tilings (XX)

Background

Informally, a Tiling Problem consists of a collection of square tiles with colored edges. We want to know whether it is possible to place the tiles on an infinite chess board in a way that the colors of adjacent edges match, filling up the whole board in the process. The tiles cannot be rotated or reflected, only translated; we assume an unlimited supply of tiles of each type. This infinite version of the problem is undecidable; the proof rests on encodings of a computation of a Turing machine and is quite messy (the original proof used a tile set of cardinality 20,426).

To push things down to $\mathbb{NP}$, we restrict our tilings to a board of size $n \times n$. Technically, we have a finite set C of colors and a tile set $T \subseteq C^4$. A tiling is now a placement of n^2 tiles so that adjacent tiles share the same color. In the anchored square tiling problem (ASTP) we are given an instance $\langle C, T, 0^n, t_1, t_2\rangle$: the colors, the tiles, the grid size n (coded in unary as 0^n) and two anchor tiles. Tile t_1 must be placed in the North-West corner of the grid, and t_2 in the South-West corner.

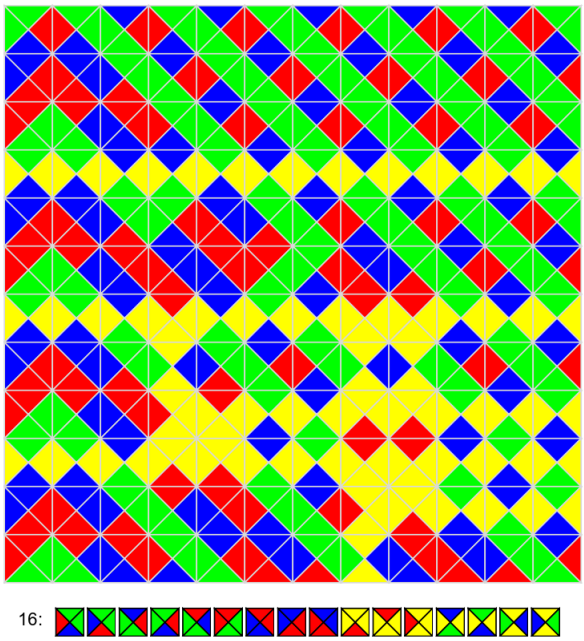
Task

A. Show that ASTP is in $\mathbb{NP}$.

B. Show that ASTP is $\mathbb{NP}$-hard by a direct simulation of polynomial time Turing machines.

C. Explain how to get rid of the South-West anchor tile, without affecting hardness.

Comment Here is an example of a 12×12 tiling using 16 tiles with colors red, green, blue and yellow.



Comment For the simulation in part (B), think of row k in the tiling as corresponding to the configuration of the machine at time k. This won't quite work out, you will need several rows to simulate a single step of the Turing machine. Also, you may want to make some harmless assumptions about the machine to simplify the construction.

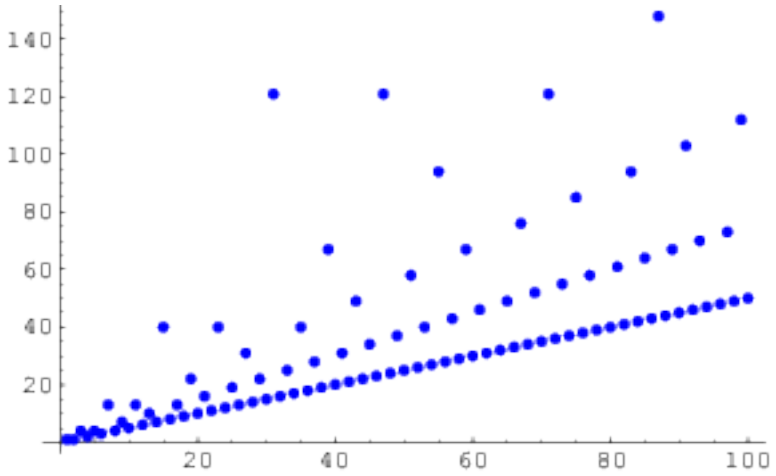
0.11. Son of Collatz (XX)

Background

A function defined by an apparently simple arithmetic operation can behave in a rather unpredictable way under iteration, the Collatz function being a prime example. Here is an example that looks somewhat similar to the Collatz function, but this one is based on recursion. Define a function F on the positive integers by

$$F(x) = \begin{cases} x/2 & \text{if } x \text{ is even,} \\ F(F(3x+1)) & \text{otherwise.} \end{cases}$$

Note the double application of F in the odd case. It is not really clear that this is well-defined, there might be some infinite loop – but there isn't. Here is a plot of F on up to $x = 100$.



Task

A. Determine what the lines in the picture are. More precisely, determine a simple description of the x-coordinates of all the points belonging to a single one of these lines (the y-coordinates are then easy to get).

B. Give a reasonably simple non-recursive description of F.

C. Prove that your description is correct, and conclude that F is really well-defined: for any positive integer x there is exactly one y such that $F(x) = y$.

D. Define $d(x)$ to be the number of recursive calls made in the computation of $F(x)$. For example, for all even x, $d(x) = 0$, $d(1) = 2$ and $d(3) = 4$. Find a simple description of d.

Comment

It is a good idea to try to figure out how this function is related to the Collatz function C. A little experimental computation might also be helpful. If you think that part (A) is a bit vague you are quite right. This builds character.

0.12. The DASZ Operator (XX)

Background

For this problem, consider non-decreasing lists of positive integers $A = (a_1, a_2, \ldots, a_w)$. We transform any such list into a new one according to the following simple recipe:

- Subtract 1 from all elements.
- Append the length of the list as a new element.
- Sort the list.

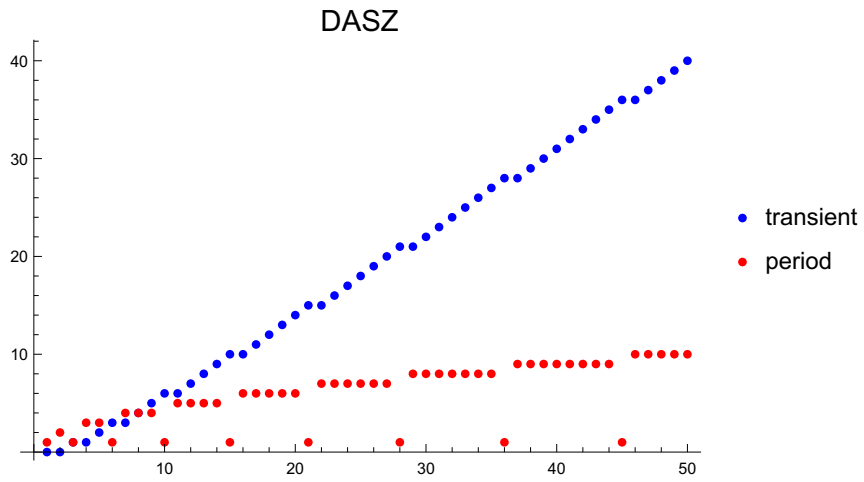
- Remove all 0 entries.

We will call this the DASZ operation (decrement, append, sort, kill zero) and write $D(A)$ for the new list (note that D really is a function). For example, $D(1,3,5) = (2,3,4)$, $D(4) = (1,3)$ and $D(1,1,1,1) = (4)$.

A single application of D is not too fascinating, but things become interesting when we iterate the operation: as it turns out, $D^t(A)$ always has a finite transient (and period), no matter how A is chosen. For example, the transient and period of $(1,1,1,1,1)$ are both 3:

0	1	1	1	1	1
1	5				
2	1	4			
3	2	3			
4	1	2	2		
5	1	1	3		
6	2	3			

Here is a plot of the transients and periods of all starting lists $A = (n)$ for $n \leq 50$.



Note the fixed points $D(A) = A$, the few red dots at the bottom.

Task

A. Show that all transients must be finite.

B. Characterize all the fixed points of the DASZ operation.

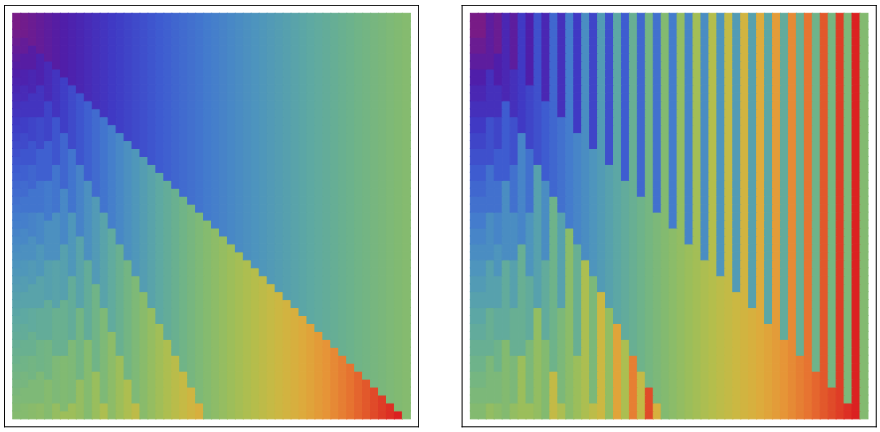
C. Determine which initial lists $A = (n)$ lead to a fixed point.

0.13. Floyd on Steroids (XX)

Background

Recall Floyd's cycle finding algorithm that allows one to compute the period of a point under some function $f : A \to A$ where A is a finite ground set. In the classical version, the two pebbles move at speeds 1 and 2, respectively. Let us refer to this as the $(1, 2)$ version of the algorithm; correspondingly, we can also consider a (u, v) version of the algorithm for other natural numbers u and v. Naturally one wonders whether other speeds could be used to find a point on the limit cycle.

Define the Floyd-time of (τ, π, u, v) to be the time when the speed u and v pebbles meet on a lasso with transient τ and period π. Here is a picture of the Floyd-times for $u = 1$ and $v = 2$ on the left, and $u = 2$, $v = 4$ on the right; $\tau, \pi \leq 50$.



Task

A. Does the $(2, 3)$ algorithm still work for all possible inputs? Justify your answer.

B. Characterize the values of u and v for which the (u, v) algorithm works (of course, works here means that it finds a point on the limit cycle for all possible inputs).

C. Find an algebraic way to determine the Floyd-time. Does your solution explain the much more complicated Floyd-times for $v - u > 1$?

D. Produce a simple closed form for the Floyd-time when $v = u + 1$.

E. Suppose the (u, v) algorithm works. How does the running time compare to the standard version? By running time we mean the number of applications of f.

Comment

A little computational experimentation might be a good idea; it's a bit tricky to get a completely correct answer.

For part (C) I suspect there is no easy closed form in general, you will need to write a little program using modular arithmetic.

Here is a helpful function, a generalization of the ordinary mod function.

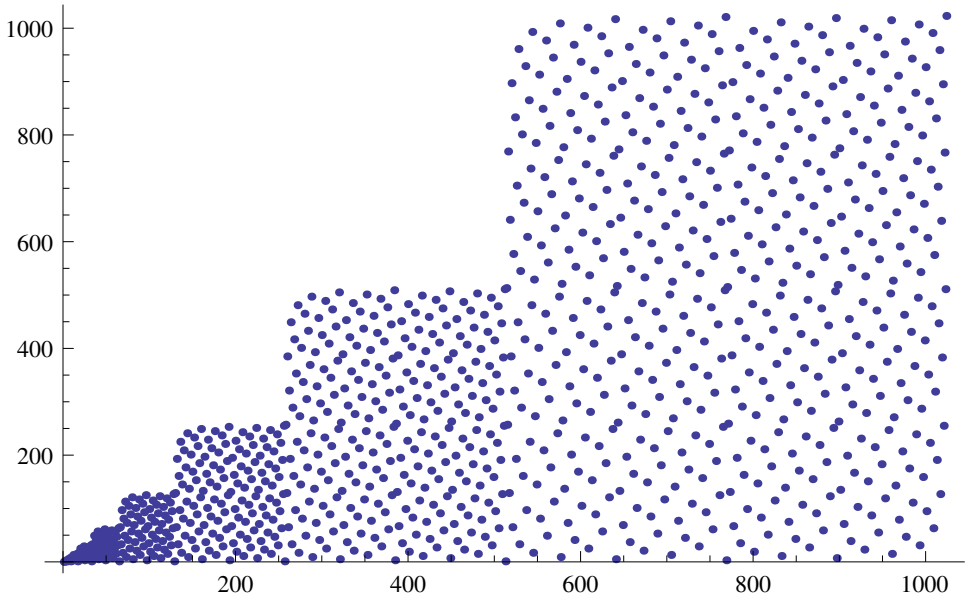
$$t \bmod \tau{:}\pi = \begin{cases} t & \text{if } t < \tau, \\ \tau + (t - \tau \bmod \pi) & \text{otherwise.} \end{cases}$$

Thus, $t \bmod \tau{:}\pi$ describes the position of a speed-one pebble at time t on a lasso with transient τ and period π. Here we assume that the lasso uses carrier set $\{0, 1, \ldots, \tau, \ldots, \tau + \pi - 1\}$.

0.14. Reversing Digits (XX)

Background

For any number x, let $R(x)$ be the number obtained by reversing the sequence of digits in the binary expansion of x, without leading 0's. For example, $R(6) = 3$, $R(8) = 1$ and $R(15) = 15$. Here is a picture of the first values of R for $x < 2^{10}$.



Make sure you understand the staircase in this picture.

Task

A. Show that for any x we have one of the following four situations:

$$R(x) = x \qquad R^2(x) = x \qquad R^2(x) = R(x) \qquad R^3(x) = R(x)$$

B. Show that a number x is divisible by 3 iff $R(x)$ is divisible by 3.

C. Give a purely algebraic proof of part (B).

D. Give yet another proof of part (B) based on the idea of scanning the binary digits from MSD to LSD and keeping track of remainders.

Comment

0.15. Flipping Pebbles (XX)

Background

An interesting class of problems deals with the question whether a certain system can ever reach a particular state, given a collection of possible transitions the system can make.

As an example, consider a collection of tokens, blue on one side, and red on the other. We have $2n$ tokens aligned on a $2 \times n$ grid, initially all tokens are blue side up. The admissible transitions are described as follows:

> Pick an arbitrary token, and flip it over; also, flip over its two neighbors.

Can the system reach the all-red state?

Task

A. Show that order of the flip operations does not matter.

B. Show that flipping the same token twice undoes the first flip, and so two flips have no effect.

C. Show that all $2 \times n$ grids of tokens can reach the all-red state by flipping tokens and their neighbors, starting from all-blue.

D. (Extra Credit) Repeat for $3 \times n$ grids.

Comment This actually works for arbitrary $m \times n$ grids, but the proof is much harder.

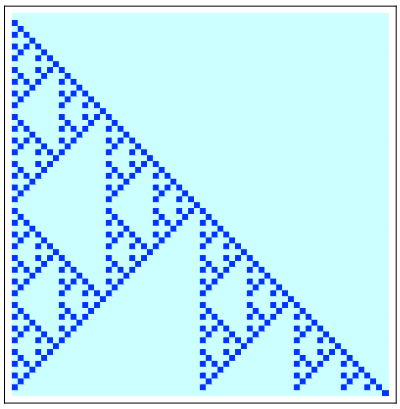
0.16. Fibonacci Polynomials mod 2 (XX)

Background

We will work in the polynomial ring $\mathbb{F}_2[x]$, where $\mathbb{F}_2$ denotes the two-element field. We can define the Fibonacci polynomials as usual by

$$\begin{aligned}\pi_0(x) &= 0\\ \pi_1(x) &= 1\\ \pi_n(x) &= x\,\pi_{n-1}(x) + \pi_{n-2}(x)\end{aligned}$$

We write c_{nk} for the coefficent of x^k in π_n, $k < n$. Here are the coefficents of these polynomials up to $n = 64$.



Write $\mathsf{bin}(n, k)$ for the kth binary digit of n. According to a theorem by Lucas, $\binom{a}{b}$ is odd iff $\mathsf{bin}(b, k) \leq \mathsf{bin}(a, k)$.

Task

A. Give a recursive description of c_{nk} and exploit it to write these coefficents in terms of binomial coefficents.

B. Exploit Lucas' theorem to produce a linear time algorithm to compute π_n (at least with uniform cost function).

C. Show that for $m < n$ we have $\pi_n = \pi_{m+1}\pi_{n-m} + \pi_m\pi_{n-m-1}$.

D. Conclude that $m \mid n$ iff $\pi_m \mid \pi_n$.

Comment

0.17. Rotations (XX)

Background

Let Σ be an alphabet of size n, and X the language of all words that are permutations of Σ. The symmetric group $\mathfrak{S}_n$ acts naturally on X on the right as

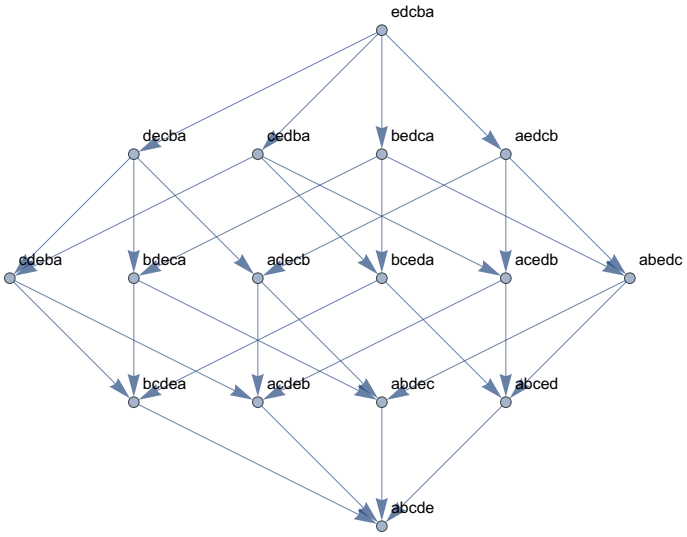
$$x \cdot f = (x_{f^{-1}(1)}, \ldots, x_{f^{-1}(n)})$$

If you find the f^{-1} irritating, use the corresponding left action instead. Clearly, the action is transitive.

Given a set Γ of generators of $\mathfrak{S}_n$ and $f \in \mathfrak{S}_n$, define the word length of f over Γ to be the least k such that $f = g_1 g_2 \ldots g_k$ where $g_i \in \Gamma$, in symbols $\mathsf{wl}_\Gamma(f)$. The word length of Γ is the maximum of the all the $\mathsf{wl}_\Gamma(f)$ for $f \in \mathfrak{S}_n$.

In the following, we will be using a redundant set Γ of generators based on right rotation: in cycle notation, $g_{ij} = (i, i+1, \ldots, j)$. Thus, g_{ij} has the effect of cyclically rotating the block $x_i, x_{i+1}, \ldots, x_j$ in x, the rest of the word is unchanged. Γ has cardinality $n(n-1)/2$.

Below is a picture of all possible short rotation sequences turning *edcba* into *abcde*.



Task

A. Show that the word length of Γ is $n-1$. Hint: look at the chain of subgroups defined by $G_k = \mathsf{Stab}_{k+1,\ldots,n}$.

B. Find a polynomial time algorithm to compute $\mathsf{wl}(x)$ given $x \in X$.

C. Now consider the problem of finding the shortest sequences of generators such that $x \cdot g_1 \ldots g_k = y$.

Comment

There is a trade-off between the size of a generating set and the corresponding word length: our example is $n(n-1)/2$ versus $n-1$. With more effort one can also produce $n-1$ versus $n(n-1)/2$; and even more effort produces $n \log n$ versus $2n \log n$.

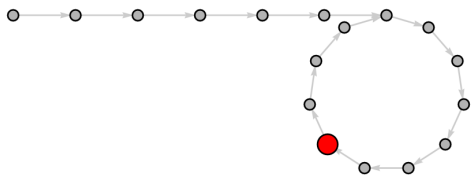
You can think of the rotations as "super-bubble-sort": instead of using only transpositions, we can use arbitrary right rotations.

0.18. Floyd goes Algebraic (XX)

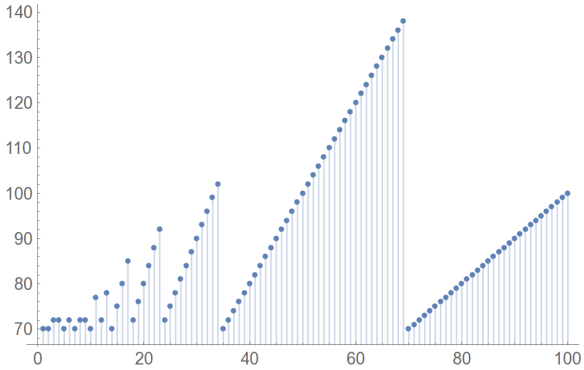
Background

One of the most elementary results about finite semigroups is that every element in the semigroup has an idempotent power. Recall that x is idempotent if $x^2 = x$. Then the claim is that for S any finite semigroup and $a \in S$, there exists some integer $r \geq 1$ such that a^r is idempotent.

Ignoring algebraic aspects for a moment, note that the powers of a (the points a^i, $i \geq 1$) must form a lasso since S is finite. Hence we can associate a with a transient $t = t(a) \geq 0$ and a period $p = p(a) \geq 1$.



We can think of the powers of a as the orbit of a under the map $x \mapsto ax$, so we can apply Floyd's trick to find a point on the loop (the big, red dot above). Let's call the time when the algorithm finds this point the Floyd time $\tau(t,p)$, an integer in the range 0 to $t+p-1$. The next picture shows the Floyd times for $t = 70$ and $p = 1, \ldots, 100$. The red dots indicate the values of p where the Floyd time is $t = 70$.



Inquisitive minds will wonder if there is any connection between the Floyd time and the idempotent from above.

Task

A. Prove the claim about idempotents in finite semigroups. Hint: think about the Floyd time.

B. Show that there is exactly one idempotent among the powers of a.

C. Give a simple description of the Floyd time $\tau(t, p)$ in terms of the transient t and period p.

D. Explain the plot of the Floyd times above.

E. Show that the powers of a that lie on the loop form a group with the idempotent as identity.

Comment

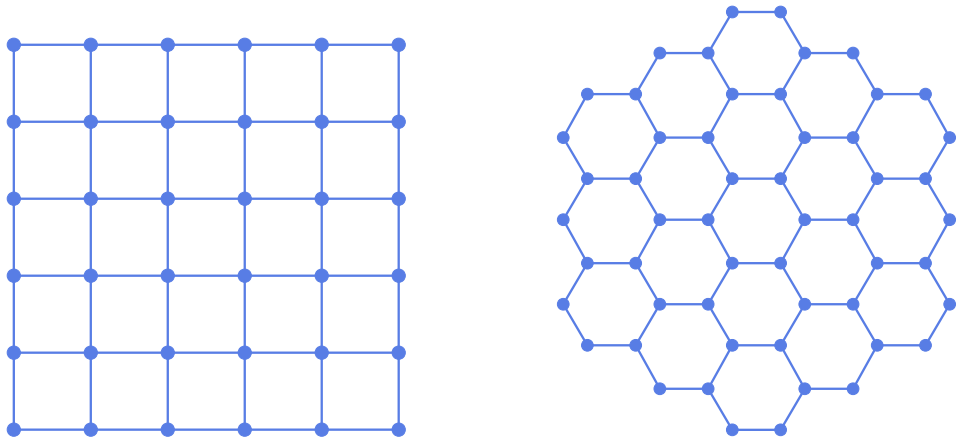
For extra credit you might (re-)consider the question of what happens in Floyd's algorithm when one changes the velocities of the particles to $1 \leq u < v$.

0.19. Grid Actions (XX)

Background

Dihedral groups describe rigid motions (rotations plus reflections) of regular polygons in the plane that return the polygon to its original position. This carries over to more complicated grids, but with the caveat that the dihedral group needs to be embedded into a permutation group of appropriate degree (the number of nodes in the grid).

We will consider the grids below:



the plain square grid SG_6 and the hexagonal grid $\mathsf{HG}_{3,3,3}$.

Task

A. Determine the automorphism group of SG_6, find generators and determine the cycle structure of the group elements.

B. Compute the orbits of the vertices under the group action. What are the orbit sizes for SG_n?

C. Compute the orbits of the edges under the group action. What are the orbit sizes for SG_n?

D. Repeat for $\mathsf{HG}_{3,3,3}$ instead of SG_6.

Comment

You probably don't want to do this by hand.

0.20. Building Reversible Cellular Automata (XX)

Background

For applications in cryptography and also in physical simulations (non-dissipative systems) it is desirable to be able to construct injective cellular automata. There is an old trick due to Fredkin that constructs an injective automaton from a given arbitrary one. Here is the idea, you will have to supply the details. Suppose ρ is the local map of an arbitrary binary cellular automaton. The orbits produced by ρ are of the form

$$X_{n+1} = \mathsf{G}_\rho(X_n) = \mathsf{G}_\rho^n(X_0).$$

For arbitrary, irreversible ρ there is no way in general to reconstruct X_n from X_{n+1}. But now consider

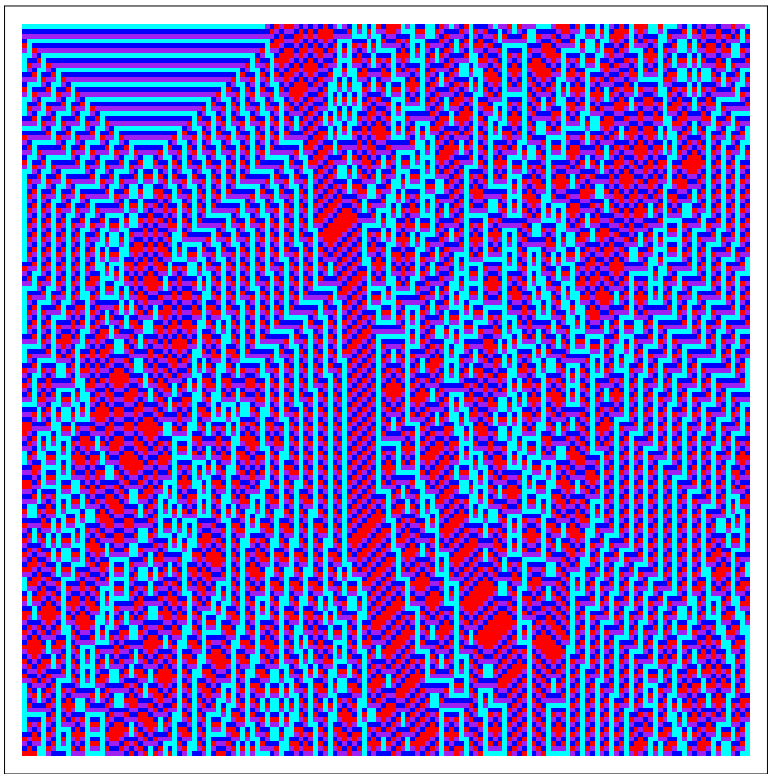
$$X_{n+1} = \mathsf{G}_\rho(X_n) \oplus X_{n-1}$$

where $\oplus$ is exclusive 'or' (or addition mod 2), as usual. Then

$$X_{n-1} = \mathsf{G}_\rho(X_n) \oplus X_{n+1}$$

and we can go backward – though we need two consecutive configurations to obtain the previous one. This also means we have to have two starting configurations X_0 and X_1.

So far, all we have is a second-order automaton, but we really want an ordinary injective cellular automaton ρ' based on ρ. To this end, we have to combine two binary configurations into a single configuration over a larger alphabet. Here is the result of this construction applied to ECA 77 (this picture really requires colors, there should by cyan, blue, purple and red).



With a bit of imagination you can almost see how this CA is reversible (there is no loss of information from time t to time $t+1$), but pictures can be very deceptive.

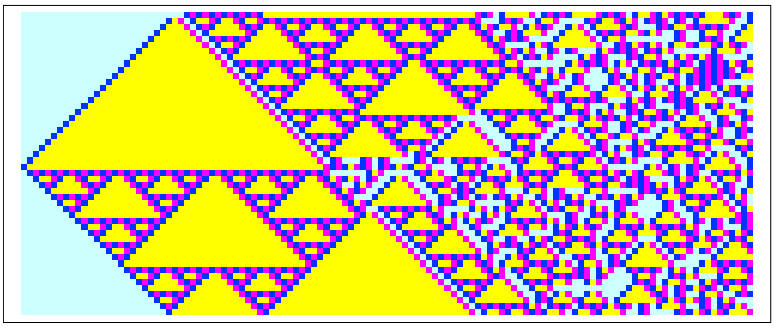
Task

1. Explain exactly how the Fredkin construction of ρ' from ρ works.
2. Prove that your automata ρ' are indeed injective, regardless of the properties of ρ.
3. Let ρ be the CA that is obtained by applying the Fredkin construction to the trivial ECA with number 0. Analyze the behavior of ρ (it's quite boring).

Comment

Needless to say, if you start from an arbitrary ECA ρ and apply Fredkin's trick things become quite interesting. For example, the pseudo-random rule ECA 30 is not reversible, but if we use it to construct a reversible rule we get the following behavior on a one-point seed: first things are fairly regular, but then the patterns become chaotic.

The first 120 steps for $n = 50$.



Incidentally, for $n = 20, 21, 22, 23, 24$ the periods are $85,044$; $887,258$; $2,217,902$; $381,601$ and $15,588,247$, respectively.