15-354: Computational Discrete Math — K. Sutner

CDM Projects — Fall 2023

Proposal Deadline

Email me a brief description of your choice of project and some preliminary thoughts about what you are going to do by **Monday, November 27**, midnight as usual.

If you plan to partner with someone, submit just one proposal and place a corresponding note into the other directories.

AFS Directories

All submissions will be to directories

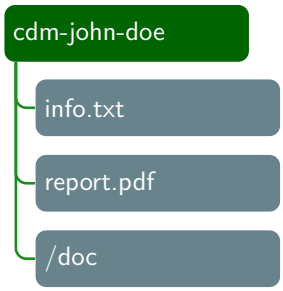
`/afs/andrew.cmu.edu/course/15/354/handin/<andrew-id>`

There is plenty of space, but do not submit gigabytes of data.

What To Do

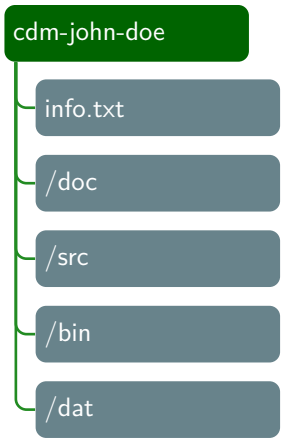
The official deadline for all projects is the last day of classes, **December 8, 24:00**. For potential extensions, talk to me. Place you tgz file into the AFS directory above, see below for specs. Your submission will contain multiple files, possibly including pdf, code, sample data files, and so on—make sure that everything is nicely organized. Submit a file `cdm-john-doe.tgz` and include a text file `info.txt` that explains the file structure of your submission (and anything else you want me to know). If there is code, make sure it compiles under a standard Andrew linux environment–no compile, no credit. I am not going to edit code in order to get it to compile, and I am not going to install special software to get things to work.

If you project is a paper report, your `tgz` file should unpack like so:



Supporting documents (and in particular the **paper under review**) go into `/doc`.

If your project involves code, adhere to the following subdirectory structure. Make sure to provide compelling examples in `/dat`, if it is not obvious how to construct more examples I will assume that your code fails in general. Your `.tgz` file must unpack like this:



Put a ready-made executable into `/bin`, just in case compilation somehow fails.

Some of the projects below can be handled essentially without recourse to the literature, others involve some reading. Do not procrastinate.

There is no need to prove new results. Of course, it's fantastic if you do come up with something new, but it is **not** required. You are expected to demonstrate a clear understanding of the issues and the ability to describe the problem and its solution.

For some of the larger projects you may think about pairing up with someone. For example, the universal Turing machine problem and the Ehrenfeucht-Mycielsky sequence are best handled with a partner.

If you have a proposal for an alternate topic, make sure to talk to me before you start working–not all topics are suitable.

As you will see, the projects have inherently different levels of difficulty; obviously, if you choose a simple project, you will be expected to produce a particularly nice solution. On the other hand, if you go after one of the hard questions, don't worry if there is no breakthrough, just do the best you can (welcome to the world of research).

If you have any question, talk to us, don't just start out on a wild goose chase.

Paper Reports

For a report, you can read any of the papers I mentioned in class; for example, the Elgot and Mezei paper on transducers, the Rabin/Scott classic, Kleene's paper on regular events. Take a look at projects. Explain the central ideas and contributions in the paper. In particular, explain the proofs and, wherever appropriate, provide examples for the given constructions. Think of this like being on a program committee: someone who reads your report should have a good idea of what is going on in the paper, and should be well-prepared to read and understand the actual paper.

In addition to the papers mentioned in class, you can also pick a recent research paper. To keep things under control, let us limit the range of papers to conference proceedings of ICALP, CIAA, DCFS, LATA, FOCS and STACS for the last 8 years. Not all the papers here relate directly to CDM, pick one that does. If you have some other paper in mind, talk to me asap.

These projects are solo exercises, pick a paper that is manageable. If there is a clash (highly unlikely), it will be resolved on first-come first-serve basis. It is a good idea to start looking for a suitable paper right now, and spend a bit of time making a good choice.

Your paper should roughly be organized like so:

- Introduction: what is the topic, why is it interesting, what is the state of the art.
- Contribution: what is the specific contribution of this paper.
- Theorems & Proofs: what are the technical results and how are they established, in particular: is everything correct?
- Critique: is the paper well-written, could it be easily improved (e.g. by including examples), are the computational parts reproducible, is the bibliography sufficient?

Take item (3) seriously, just because a result is published does not necessarily mean that it is quite correct. After reading your analysis of the paper, it should be quite straightforward for a mathematically literate reader to go through the actual paper.

Specific Projects

- Universal Register Machine

  Construct a complete URM, without using the macros from lecture. You might want to write a small compiler that translates the version using macros into a real register machine. Try to make the resulting machine as small as possible and demonstrate its correctness by simulating a few small register machines.

  To make interesting simulations feasible, you need to write an "intelligent" interpreter: analyze cycles in the diagram of the machine and speed up their execution by performing giant-steps rather than baby-steps.

  The deliverable must be useful for demonstrations: I want to be able to show an actual universal computer perform simple tasks such as addition or perhaps multiplication.

  Your code must run on a standard Andrew linux box, specialized software is not acceptable.

- One Instruction Language

  Universal computers can be obtained in many ways. One possible approach is a "one instruction language (OIL)." Show that the classical OIL is computationally complete. Write an interpreter for OIL programs and construct some interesting examples, see OneInstruction The point here is that your system must produce convincing examples (use a smart interpreter rather than the obvious brute-force one), there is nothing challenging about finding some arbitrary OIL.

- A Small "Universal" Turing Machine

  There is a claim that a 2-state/3-symbols Turing machine can already be universal, seef

  - 2-3-Turing Machine,
  - 2-3-TM Prize,
  - 2-3-TM Proof.

  As you will see, the alleged proof by Alex Smith has lots and lots of holes, see for example the objections raised at FOM linked at the wiki site.

  The ultimate goal here is to determine whether Smith's argument holds water. This is probably too hard, so any significant progress towards a resolution will count. For example, a clean and mathematically precise version of the construction would already be of interest.

- Divisibility and Minimal DFAs

  Determine the size of the minimal DFA recognizing all multiples of a fixed modulus m in various numeration systems. This has been done for normal radix notation and reverse radix, but the results can probably be improved upon, see Divisibility.

- BDDs and Finite State Machines

  Use a freely available ROBDD package such as CUDD to implement finite state machines. More precisely, implement NFAs and DFAs using ROBDDs, and a determinization algorithm. Your algorithm should cope with large DFAs that are difficult to handle using more traditional data structures. CUDD is quite a handful, alternatively you can use the ROBDD implementation in Mathematica.

  In particular you should be able to deal with families of NFAs that demonstrate exponential blow-up during determinization such as the "kth symbol from the end" automata.

- Polya-Redfield Implementation

  There are lots of algorithmic questions surrounding the Polya-Redfield method, see for example an email I received regarding a previous version of the course: Halbersma. As it turns out, Doron Zeilberger

has written a large Maple package that deals with some of these questions, see Graph Enumeration. Translate this package (or at least a substantial part of it) into Mathematica and give examples. There is also an old package in Mathematica that I will provide.

- SAT Solver

  The Satisfiability problem for propositional logic (rather than first-order logic) is obviously decidable, though it is currently no known whether it can be handled in polynomial time. Implement a simple SAT solver along the lines of the classical Davis/Putnam algorithm DPLL. The point here is not blinding speed but easy traceability. For example, it should be easy to demonstrate the effect of various heuristics in the splitting case.

  You can check your algorithm against one of the performers in the SAT competition SAT Competition. This is(for correctness only, not for speed; the latter will only lead to depression on your part.

- Cycle Finding Algorithms

  Floyd's algorithm is just one example of a memoryless cycle finding algorithm. Implement a number of these algorithms and compare their performance. A good test case are the 256 elementary cellular automata. Try to compute transients and periods for n-bit configurations for n as large as possible and try to classify ECAs accordingly. Long transients and periods are particularly interesting (e.g., the pseudo-random number generator ECA 30 should have very long periods).

- Testing for k-Cycles

  k-cycles are obviously a first-order property (for fixed k), so we can decide their existence by finite state machines over automatic structures. Consider the concrete case of elementary cellular automata operating on finite configurations, assuming periodic boundary conditions. Implement an algorithm that, given the ECA number r, $0 \leq r < 256$ and $k \geq 1$, determines for which $n \geq 1$ the ECA has a k-cycle on $\mathbf{2}^n$. The challenge here is to handle "large" values of k up to, say, 20.

- Busy Beaver

  Read the paper Harland to see what is currently wrong with attempts to determine some particular values of the BB function. Come up with a realistic framework to do better in the future–perhaps something like a BB-github. Explain how your system would guarantee an improvement over the current situation.

- N. J. Wildberger

  Wildberger is a mathematician in Australia, who has a few rather unorthodox ideas. E.g., he does not believe in the fundamental theorem of algebra Wildberger (which he likes to call the "fundamental dream of algebra").

  Look at some of his more unconventional claims, and write a critique. It might help to compare Wildberger's work to Doron Zeilberger at Rutgers. Zeilberger's recent "What is Mathematics and What Should it Be?" is particularly wrongheaded and annoying.

- Ehrenfeucht-Mycielsky Sequence

  There is a simple, recursively defined ω-word over the alphabet $\{0, 1\}$ that apparently has limiting density $1/2$. In fact, computing a few million bits of the sequences shows very rapid convergence. Unfortunately, no proof is known though lots of little facts have been unearthed by a number of people, see EM-sequence. and search the web.

  Make a dent in this problem, find something new and interesting. A full proof would be nice but is probably quite hard.

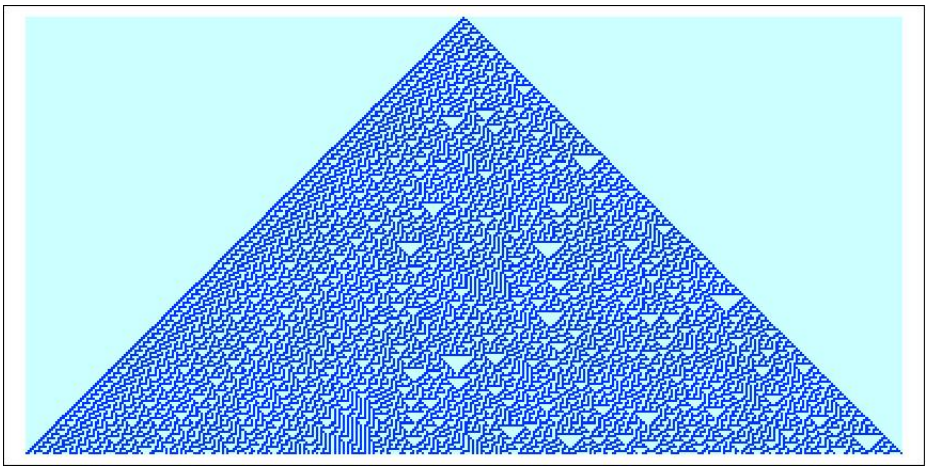
- Dropping Balls

There is an old single-player game dating back to the 1960s that has a nice explanation in terms of group actions, see Trellis Automata.

The first three questions in the reference are quite manageable, but the last one (generalize to other trellises) is utterly open-ended.

- ECA 30

  Elementary cellular automaton number 30 has amazing properties, it seems to generate random patterns.



You can now earn real money by solving some questions related to ECA 30, see ECA 30. This is probably too hard, but if you think you might want to take a swipe at it, talk to me first (I am on the prize committee).