

Computational Discrete Mathematics

Klaus Sutner
Carnegie Mellon University

Fall 2011

Outline

- 1 A Brief History of Computation
- 2 CDM Philosophy (Ideology?)
- 3 Administrivia
- 4 Not So Frivolous Example

Ancient Mathematical History

For our purposes, that's everything up to around 1800.

Up to this point, mathematics was doing just fine: there were generally accepted modes of mathematical reasoning that were considered to be perfectly reliable—assuming the practitioner did not blunder.

To the modern eye, much of the reasoning would seem hair-raisingly informal, just take a look at Euler and Fourier.

19th Century

Things started to crumble in the 1800's.

If Gauss says he has proved something, it seems very probable to me; if Cauchy says so, it is about as likely as not; if Dirichlet says so, it is certain.

C. G. J. Jacobi, in a letter to A. von Humboldt

Hitherto accepted modes of mathematical reasoning were recognized as being much less reliable than one had assumed.

The problem was that notions that seemed intuitively clear (such as continuity) actually required a lot of attention, otherwise errors were quite possible (even super-stars like Cauchy did occasionally founder due to conceptual vagueness).

The Antidote

Apparently, the only reliable solution to this problem was to be exceedingly formal and precise in all arguments. At least three ideas emerged that appeared to be helpful in this enterprise:

- Symbolic logic (Frege)
- Axiomatization (Peano, Dedekind)
- Set theory (Cantor)

The first two merged more or less into the notion of a **formal system** and set theory developed into the de facto gold standard (Bourbaki).

Russell and Whitehead

As it turns out, it is easy to design a formal system that is inconsistent. For example, Frege had an extremely elegant and concise system that fell prey to a paradox discovered by Russell.

To construct a formal system that avoids inconsistencies and is also powerful enough to cover all of mathematics is rather difficult. The approach taken by Russell and Whitehead in their *Principia Mathematica* is horribly technical and was never appreciated by “ordinary” mathematicians.

To add insult to injury, Gödel showed some 20 years later that any system like Principia is necessarily incomplete: some true statements cannot be proven in the system. In particular consistency is most elusive.

Zermelo-Fraenkel Set Theory

A technically less daunting approach is to *axiomatize* set theory.

Zermelo proposes a system in 1908, augmented in 1922 by Fraenkel. ZF set theory has only 9 simple axioms (there are variants):

extensionality, empty set, unordered pair, union, power set, separation, replacement, foundation, infinity.

Hugely successful, still the de facto gold standard in mathematics (Bourbaki) and TCS.

Hilbert's Program

Formalize mathematics and concoct a finite set of axioms that are strong enough to prove all theorems of mathematics (completeness) and show that the system is consistent; by strictly finitary means. Also show that statements about “ideal objects” can be proven in the system, without using ideal objects. Lastly, come up with a decision algorithm for mathematics.

Initially some good progress (completeness of propositional logic, then of predicate logic).

But then in 1931 Gödel drops a bombshell: any mathematical system built on predicate logic is necessarily incomplete (or inconsistent).

Computability

Though some parts of Hilbert's program were irreparably damaged by Gödel's result, in many ways things just started to become really interesting.

Thus the notion "computable" is in a certain sense "absolute," while almost all metamathematical notions otherwise known (for example, provable, definable, and so on) quite essentially depend upon the system adopted.

K. Gödel, 1936

There is a clear connection between incompleteness and unsolvability, so computability is a rather central notion in mathematics.

Entscheidungsproblem

The Entscheidungsproblem is solved when one knows a procedure by which one can decide in a finite number of operations whether a given logical expression is generally valid or is satisfiable. The solution of the Entscheidungsproblem is of fundamental importance for the theory of all fields, the theorems of which are at all capable of logical development from finitely many axioms.

*D. Hilbert, W. Ackermann
Grundzüge der theoretischen Logik, 1928*

In modern terminology: find a *decision algorithm* for statements of mathematics (or at least some part like arithmetic, group theory, ...).

Recursion Theory

The early development of the theory of computation (historically referred to as recursion theory) took place before the development of a actual, physical computers – though Alan Turing and John von Neumann made substantial contributions in both fields.

As a consequence, many of the central notions of recursion theory are not at all concerned with practical computation. For example, fine-grained resource bounds such as polynomial running time play no role at all.

Parts of the theory are technically daunting and seem quite far removed from any connection to actual computation.

Hao Wang

The study of degrees [of unsolvability] seems to be appealing only to some special kind of temperament since the results seem to go into many different directions. Methods of proof are emphasized to the extent that the main interest in this area is said to be not so much the conclusions proved as the elaborate methods of proof.

John von Neumann

Everybody who has worked in formal logic will confirm that it is one of the technically most refractory parts of mathematics. The reason for this is that it deals with rigid, all-or-none concepts, and has very little contact with the continuous concept of the real or of the complex number, that is, with mathematical analysis. Yet analysis is the technically most successful and best-elaborated part of mathematics. Thus formal logic is, by the nature of its approach, cut off from the best cultivated portions of mathematics, and forced onto the most difficult part of the mathematical terrain, into combinatorics.

- A Brief History of Computation

2 CDM Philosophy (Ideology?)

- Administrivia
- Not So Frivolous Example

... but there is help



Alan Turing

"I expect that digital computing machines will eventually stimulate a considerable interest in symbolic logic . . . The language in which one communicates with these machines . . . forms a sort of symbolic logic."

William Thurston

“The standard of correctness and completeness necessary to get a computer program to work at all is a couple of orders of magnitude higher than the mathematical community’s standard of valid proofs.”

Uses of Computing in Math

- **Number Crunching**

E.g., to solve complicated differential equations. Historically the first major application; fraught with problems.

- **Symbolic Computation**

Directly manipulate symbolically presented entities (computer algebra).

- **Example Generation**

Examples are critical to understand a concept, producing them by hand can be too time consuming.

- **Counterexample Generation**

Can help in avoiding dead-end arguments in a proof.

Less Developed

- **Theorem Proving**

On rare occasions, proof assistants and automatic theorem provers are helpful in finding (parts of) a proof. Better at verification (proof checkers) than search, not easy to use.

- **Knowledge Management**

A global mathematical library would be some 10^8 pages. Some small but growing part of this is available in digital form on the web. Some even smaller part is indexed and searchable (semantic markup). Some yet smaller part is validated.

The Knowledge Soup

There is an ever increasing amount of math material on the web: MathWorld, PlanetMath, wikis, journals, research websites, . . .

Be careful, though – not all the stuff out there is reliable. Sometimes the information is misleading, and sometimes it's simply false.

Try to stick to reputable sources. And, always do a sanity check.

Computational Discrete Math

The central axiom of CDM is very simple:

- Computation helps to understand mathematics and TCS.
- Mathematics and TCS help to compute.

Duh, is a Bluebird blue?

The CDM Loop

- Specify/Formalize

We start with a question, often vague and imprecise. With some effort the question is turned into a concise and precise problem in (discrete) mathematics.

- Experiment/Compute

To help develop basic understanding we use computation to generate data (examples and counterexamples). Setting up the programs may well require a bit of work.

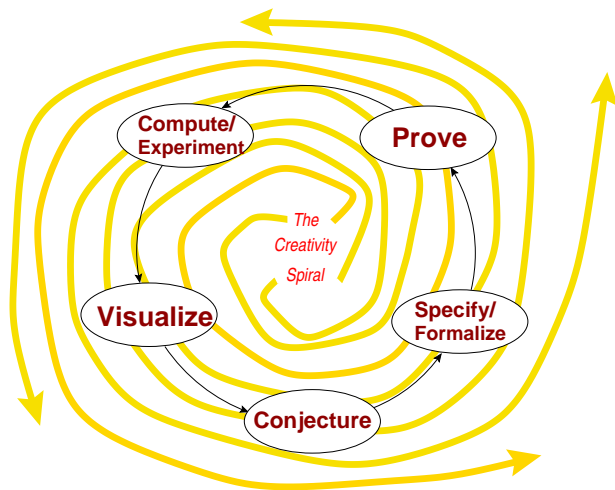
- Analyze/Visualize

Analyze and interpret the data, find patterns and structure.

The CDM Loop, 2

- Specify/Formalize
- Experiment/Compute
- Analyze/Visualize
- Conjecture/Prove
Ultimately formulate a conjecture and proceed to prove it. Wrong conjectures and dead-ends in proof attempts can sometimes be eliminated by more computation.
- Apply/Generalize
Use the new and proven theorem to improve the power of computation; discover new questions.

The Magic Spiral



Harsh Reality

In theory there is no difference between theory and practice.

In practice there is.

Yogi Berra

The Catch

In practice, there is a little issue, a resource allocation problem. You usually have a choice, you can either

- think and reason, or
- program and compute.

Sadly, this is often an exclusive or.

This is not to say that programming is mindless (quite the opposite), but it is very different from classical, paper&pencil based reasoning.

The Nature of Proofs

A proof only becomes a proof after the social act of “accepting it as a proof”.

Yuri Manin

We are not dealing with formal, computer-checked proofs; no one is except in a handful of cases.

It is important to get a good intuitive feel for when a proof is right and when it is not.

- A Brief History of Computation
- CDM Philosophy (Ideology?)
- ③ Administrivia
 - Not So Frivolous Example

Web and Communication

- Course web site:

<https://www.colormygraph.com/15354-f11/>

- Too small for fancy comm tools, let's just email.

Course Staff

- Prof:
Klaus Sutner, `sutner@cs.cmu.edu`
- TA:
TBA `tba@andrew.cmu.edu`
- Course secretary:
Rosie Battenfelder, `rosemary@cs.cmu.edu`

Course Material: Good News

- There is no text book.
- A typical Discrete Mathematics textbook is 1000 pages nowadays and mostly useful for weight lifting.
- Some references are posted on the web. Alas, none of these texts are entirely appropriate.
- I am writing a book, but only bits and pieces exist at this point.

Course Material: Bad News

- There will be a handout for each lecture.
- A lot of material will be posted on the web.
- It is a good idea to read **all this stuff**.
- The web is often helpful. Again, use with care.
- Lastly, a revolutionary suggestion: go to the library.
- And: Turn up at office hours regularly, not just at times of major crisis.

Learning Style

- The topics covered in this course translate into quite a bit of material.
- All the most important items will be presented in class, but there will necessarily be things that are left out. It is your responsibility to acquire this additional material.
- Again, read the notes, search the net, go to the library, talk to each other, talk to us.
- One of the desired outcomes of this course is that you know where to find more information should you ever need it.

Assessment

- The usual testing:
 - homeworks 50%
 - midterm (in-house) 20%
 - final 30%
- The homework is crucial in this course; solving actual problems is the only way to really get a grip on the material.
- The midterm is constrained to 80 minutes; think of it as a little reality check.
- Final means take-home final or a project. Let's talk after the midterm.

Preserving TA Sanity

- The homework will involve quite a bit of math (surprise, surprise), so it is essential that it is typeset, not hand-written.
- Submit pdf on the due date (we will either use a handin system or email).
- Do not even think about sending email.
- If you use a program in your homework make sure to reference it properly (but do not hand in 50 pages of code).

Typesetting

- There are several ways for you to generate the files:
 - \LaTeX has a bit of a learning curve, but since it is the publication standard for CS and math it's a good idea to learn how to use the system now.
 - Mathematica has a reasonable WYSIWYG editor. It's not as good as \LaTeX , but easier to use initially.
 - Anything else that can produce math.
- Make sure you choose a good, solid editor (i.e. emacs) and become an expert in your system of choice.

Lateness

Depends on what colormygraph can currently handle.

In first-order approximation, expect to be docked 10% of points for every late day (more later).

Cooperation

- Lectures will be warm and friendly. Make sure to be an active participant – CDM is not a spectator sport.
- You are strongly encouraged to talk about the course material to each other, the course staff and other students.
- This includes discussions of homework problems.

Limits to Cooperation

- However, even after ample consultation, the work you submit must be written entirely by yourself.
- List all your “consultants” on the first page of your homework.
- To avoid problems with originality, **do not take notes** when discussing homework problems.
- If you write on a board, erase everything in the end.

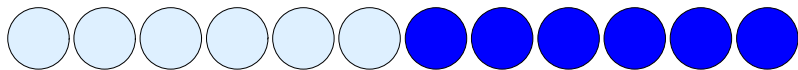
More on Limits

- Think of this as being able to copyright your solution: you may have had conversations about it with other during the problem solving phase, but the actual work is solely yours. No one should be in the room when you start writing things up.
- Needless to say, you have to be able to explain all the details of your solution at any time.
- Don't even think about copy & paste (from each other or the web), file sharing, clairvoyance, telepathy, . . .
- If you have any questions about policy issues talk to me or Alan, preferably some time before you get into trouble.

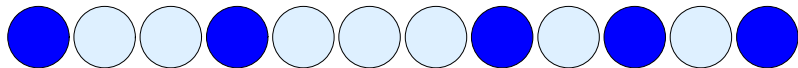
- A Brief History of Computation
- CDM Philosophy (Ideology?)
- Administrivia
- ④ Not So Frivolous Example

Flipping Pebbles

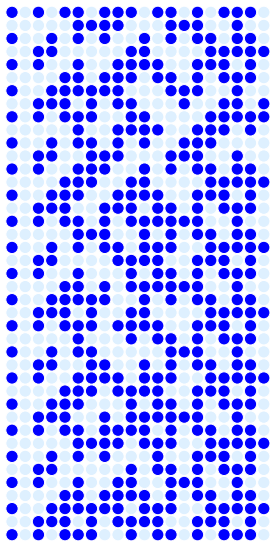
Take a row of tokens, white on one side, blue on the other.



Starting at the left, flip the current token. If it is now white, skip the next token. Otherwise, skip the next two tokens. Repeat till you fall off the end.



And Repeat . . .



Pebbles, Schmebbles

- We are really talking about the set 2^* of all (finite) binary sequences, aka binary words.
- The flipping rule defines an operation $\tau : 2^* \rightarrow 2^*$.
- We want to “understand” this operation.

Easy Observation

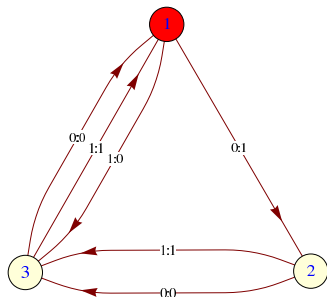
First, τ is trivially **length-preserving**.

Second, τ is **reversible**: we can reconstruct x from $\tau(x)$.

By General Abstract Nonsense (aka basic math facts) τ is just a permutation of 2^* . The operation produces cycles of words, each cycle containing only words of the same length.

Full Disclosure: Transducers

Our pebble-flipping operation is really a transduction on binary words.
The corresponding 3-state **transducer**:



We are really interested in understanding the behavior of transducers of this type.

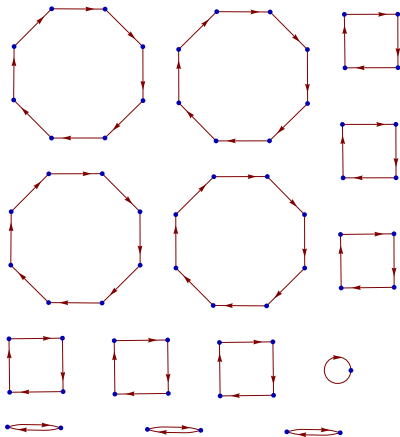
A Small Cycle

A small example:

000 → 100 → 001 → 101 → 000

So 000 produces a cycle of length 4.

Some More Cycles



Pop Quiz

- Beware beautiful pictures. Does this one make any sense?
- For example, explain the fixed point.
- Then explain the three 2-cycles.
- Overall, what might this be a picture of?

The Right Questions

- How many cycles are there?
- How long are these cycles?

More precisely, we would like answers for each word length n .

We are really talking about the **functional digraph** of τ , also called a **permutation structure** in model theory.

The Right Questions, 2

- How hard is it to test membership in a cycle?
- How hard is it to compute the least element?

These are **computational complexity** problems; note that it is important here that points are not anonymous but have internal structure.

We want elegant algorithmic answers as well as upper and lower bounds.

Need More Data

One could sit down with paper and pencil and compute a lot more cycles such as

0000 → 1001 → 0011 → 1010 → 0000

This is an excellent way to waste time and go mad in the process.

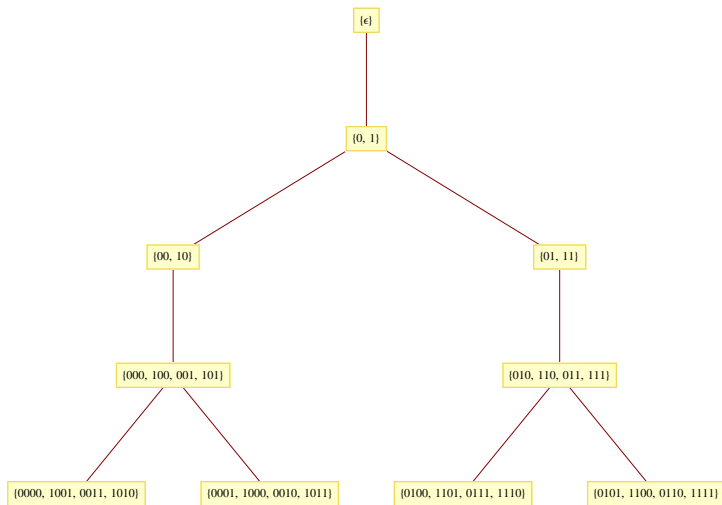
Much better is to write a little program and compute some cycles by brute force.

Cycle Count/Length Table

	1	2	4	8	16	32
0	1					
1		1				
2		2				
3			2			
4			4			
5				4		
6				8		
7					8	
8					16	
9						16
10						32

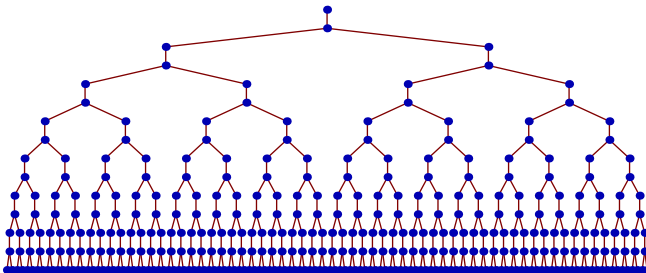
→: cycle length, ↓: word length, missing entries are 0

Cycle Tree



The root corresponds to the fixed point ε (the empty word).

Cycle Tree, Schematic



Note: the tree seems to be **regular**: there are only 2 subtrees.

Counting Cycles

Lemma

There are $2^{\lfloor k/2 \rfloor}$ cycles on words of length k .

The length of each cycle is $2^{\lceil k/2 \rceil}$.

The lemma says in essence that the picture does not lie.

There is a simple proof, but it is quite hard to find.

A Harder Question

How hard is it to test if u and v lie on the same cycle?

An obvious upper bound for a decision algorithm on words of length $2k$ is

$$O(k2^k)$$

It takes $O(k)$ steps to compute $\tau(x)$ and there are at most 2^k words to consider.

Can we do better than this? Is there a computational shortcut?

Deciding Equivalence

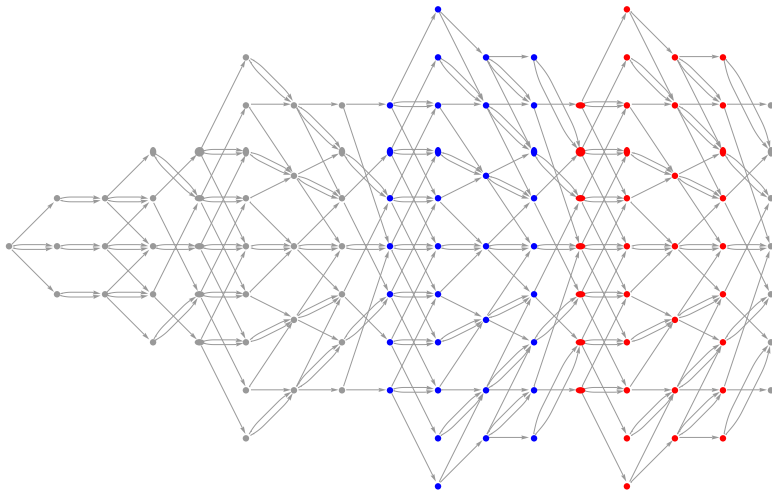
Theorem

Given $x, y \in 2^k$ one can test in linear time whether x and y lie on the same cycle (actually, a finite state machine can do it).

The proof relies on a direct construction of the machine and requires a lot of background.

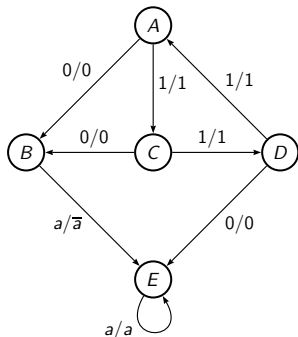
I was originally trying to prove that this is false, but my attempts to show that infinitely many states are needed produced a magic picture.

The Magic Picture



The Grigorchuk Automaton

Machines like the 3-state transducer from above are very important in group theory and symbolic dynamics.



For example, the above is a world-famous transducer and helped to solve an open problem in group theory.