UCT

Intermediate Problems

Klaus Sutner

Carnegie Mellon University

Spring 2024

Where Are We?

- We have a collection of practical $\mathbb{NP}$-complete problems.
- It is perfectly reasonable (though not uncontroversial) to assume that these problems do not admit polynomial time solutions.
- Suppose that indeed $\mathbb{P} \neq \mathbb{NP}$.
  What can we say about these two classes?

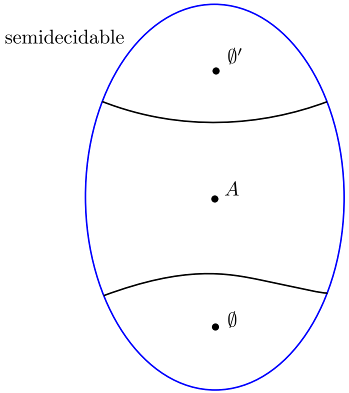
Public Service Announcement

There will be three fairly difficult proofs, the rest of the time I will wax poetic about profound matters. Enjoy.

The "proofs" will be very, very sketchy, the reason being that an actual, careful a argument would be pages and pages long, and drive everybody to distraction.

Don't worry about the endless technical details, just try to understand the general proof strategy.

Post's Question



Is this picture accurate?

Is there an intermediate semidecidable set A such that $\emptyset <_T A <_T \emptyset'$?

We can borrow an idea from classical recursion theory.

Why?

Because it is a matter of experience (not theory) that any **natural problem** that is recognized as being semidecidable ultimately will either

- turn out to be complete, or
- turn out to be decidable.

In other other words, semidecidable sets that occur in the RealWorld^{TM} only appear at two levels of complexity, bottom or top. Nothing else ever happens.

It may take a long time to determine which is the case (Diophantine equations took 70 years), but, in the end, everything turns out to be decidable or complete. Basta, no exceptions[†].

[†]As of February 2024. I'm quite confident that this won't change for a long time.

Natural?

A “natural problem” is hard to define, it means in essence: not artificially constructed by some mischievous logician.

Everybody would agree that the following would be a convincing example of a natural, intermediate problem:

> In the late 19th century, the well-known number theorist Prof. Dr. Wurzelbrunft discovered a very interesting class of integer polynomials, nowadays referred to as W-polynomials.
>
> It was shown in the 1960s that deciding whether a W-polynomial has an integer root has intermediate complexity.

Warning: This is fake news. Nothing like this has ever happened, not in any area of mathematics where undecidability results are known.

Friedberg/Mučnik and Sacks

So E. Post asked in 1944 whether there are any intermediate semidecidable sets (i.e., more than just two semidecidable degrees).

Surprisingly, two people found the solution almost simultaneously: R. M. Friedberg (an undergraduate) in the US and A. A. Mučnik in Russia (they published their results in 1957 and 1956, respectively).

Theorem (Friedberg, Mučnik 1957/6)

There are intermediate semidecidable problems.

What is perhaps even more surprising, Friedberg and Mučnik used essentially the same method (nowadays called a finite injury priority method).

Priority Arguments

This proof technique has since become the hallmark of computability theory and has been used to establish countless results.

Unfortunately, the method is somewhat complicated, in particular if one tries to seriously address all the many technical problems. At some point one becomes convinced that the intuition behind the argument is correct and gives up on serious formalization.

Surprisingly, priority style arguments are also useful in complexity theory, see Ladner's theorem below.

Density

Friedberg/Mučnik shows that there are semidecidable sets of incomparable complexity. Here is another strange result, due to G. Sacks, using a more complicated priority argument.

Theorem (Sacks, 1964)

Suppose A and B are semidecidable and $A <_T B$.
Then there is another semidecidable set C such that $A <_T C <_T B$.

Repeating this construction we can generate a dense order of semidecidable sets, like the rationals. This is not particularly intuitive, to say the least. In fact, your head might explode thinking about it.

How to Construct a Semidecidable Set

To describe the construction of some complicated semidecidable set A, it is best to use the characterization of semidecidable sets as being recursively enumerable: we use our old method of building things in stages $\sigma \in \mathbb{N}$.

At stage σ one builds a finite set $A_\sigma \subseteq \mathbb{N}$. In the end we set $A := \bigcup_{\sigma \geq 0} A_\sigma$.

Write $A_{<\sigma} = \bigcup_{\tau < \sigma} A_\tau$ for the part constructed prior to stage σ.

The key idea is that there is an easily computable (say, primitive recursive) function $\mathcal{C}$, the construction, that determines what to do at level σ:

$$A_\sigma = \mathcal{C}(\sigma, A_{<\sigma})$$

One-Way

Critical Constraint:

We can place new elements into A at stage σ, but we cannot take them out at a later stage (if we realize that something has gone wrong).

Moreover, the construction $\mathcal{C}(\sigma, A_{<\sigma})$ has to be easily computable: our decision to put an element into A cannot depend on undecidable properties.

For example, it would often be convenient to have $\emptyset'$ as an oracle, but that's not allowed. Fuggedaboud high-powered set-theoretic constructions.

The Dilemma

We are stuck with a fundamental problem:

- the set must to be semidecidable, but
- the set must satisfy complicated conditions, and typically infinitely many of them.

The solution to this problem is to use a sophisticated strategy that satisfies more and more of the conditions as σ increases. In the limit, all the conditions are ultimately satisfied and everything is fine.

A Trick

It suffices to construct two incomparable semidecidable sets, i.e., two sets A and B such that

$$A \not\leq_T B \qquad \text{and} \qquad B \not\leq_T A$$

If we succeed in doing this, then both A and B must be intermediate: neither one can be decidable, and neither one can be complete.

This may seem like a bold step in the wrong direction, but by exploiting symmetry this actually simplifies matters a bit.

Notation

We systematically confuse sets $A \subseteq \mathbb{N}$ with their characteristic functions $\mathbb{N} \to \mathbf{2}$.

So we can write things like

$$A \not\simeq \{e\}^B$$

to indicate that oracle B does not decide A via program e.

But beware: this property is **not** decidable, even if we somehow had complete knowledge of A and B. And we emphatically don't, all we have is some finite approximations $A_{<\sigma}$ and $B_{<\sigma}$.

Requirements

Or: the power of wishful thinking.

The two sets are constructed in stages, alternating between A and B in a completely symmetric manner. To make sure they have the right properties we use requirements:

(R_e)	$A \not\simeq \{e\}^B$	insure $A \not\leq_T B$
(R'_e)	$B \not\simeq \{e\}^A$	insure $B \not\leq_T A$

These are infinitely many requirements, two for each $e \in \mathbb{N}$.

Organizing Work

Since there are infinitely many requirements we have to be a bit careful about organizing our construction and use dovetailing: we work on more and more requirements as the construction unfolds.

- At stage σ we only consider objects $< \sigma$ (so there are only finitely many things to do).
- We only consider requirements (R_e) and (R'_e) for $e < \sigma$.
- And we run computations only for at most σ steps.

As a result, the construction at stage σ is quite easily computable.

And the sets A and B are indeed semidecidable.

Finite Approximations

We need to use oracles that are only partially constructed. Say, at stage σ, what could

$$\{e\}_{\sigma}^{A_{<\sigma}}(a) \simeq b \in \mathbf{2}$$

mean?

Recall that everything we touch is restricted to $< \sigma$. So we start the computation of $\{e\}$ on input a. If we get to a query $\boxed{42 \in A_{<\sigma}?}$ we do a table lookup in the finite list for $A_{<\sigma}$ and return Yes/No accordingly.

If the computation finishes in fewer than σ steps on output b we're good. Otherwise we consider the equation false.

The Catch

Suppose at the time of the query $\boxed{42 \in A_{<\sigma}?}$ the answer was No and we get some output b.

Later on it may happen that we throw 42 into A, so now the oracle says Yes, and the computation changes.

This is the crux of the whole construction, and one needs to walk on eggshells to make sure that in the end everything works out fine.

But How?

Suppose we find at stage σ that requirement (R_e) is currently broken (in the sense that the construction will fail unless we can change things at a later stage):

$$A_{<\sigma} \simeq \{e\}_{\sigma}^{B_{<\sigma}}$$

We have to break this, otherwise it might turn out in the end that $A \simeq \{e\}^B$ and A is reducible to B.

Big Question: How?

All we can do is add elements to A or B.

Witnesses

Here is what seems to be the only hope: suppose also that we discovered some witness $a = a(e, \sigma)$ for (R_e):

$$a \notin A_{<\sigma} \qquad \text{and} \qquad \{e\}_{\sigma}^{B_{<\sigma}}(a) \simeq 0$$

The next move is practically forced: throw a into A_σ, and we have broken the equality, at least for the time being.

Attention

Terminology: we say that requirement (R_e) requires attention.

Action: We throw a into A_σ.

We say that the requirement receives attention. For the time being, the requirement is happy and goes to sleep.

What could possibly go wrong?

Chaos

Say, requirement (R_e) just received attention.

But there are other requirements, say, (R'_ℓ):

$$B \not\simeq \{\ell\}^A$$

Since we just changed A, it may happen that some computations using oracle A now change, and now requirement (R'_ℓ) is angry.

We say that requirement (R_e) has just injured requirement (R'_ℓ).

And, of course, if we fix (R'_ℓ) we may clobber some other requirement, and so on. It looks like we may satisfy a few requirements, but there is no obvious way to deal with all infinitely many of them.

Priority

Here is the central idea: we prioritize requirements as in

$$R_0 > R'_0 > R_1 > R'_1 > R_2 > R'_2 > \dots$$

We will work on (R_e) at even stages, on (R'_e) at odd stages.

Basic Rule: a requirement can only receive attention (throw some witness into A or B) if that does not injure a higher priority requirement.

This has the effect that the high priority requirements get taken care of first, then the lower priority ones. In the limit, all requirements are satisfied: endless bliss.

Succeed At Last

Suppose higher priority requirement R_e blocks R'_ℓ: we would like to place b into B, but we can't.

To give R'_ℓ a fair chance, we make sure that R_e blocks only finitely many elements (the ones that could ruin the computation), and, as the construction unfolds, one considers larger and larger potential witnesses.

Ultimately, one of these witnesses is going to get through. This is a finite injury priority argument, after getting dinged finitely often all requirements succeed.

The precise proof is technically messy, but requires no more than plain induction on the naturals (rather weak in a proof theory sense).

Natural Intermediate Problems?

Alas, the Friedberg/Mučnik construction is really quite frustrating: it builds a semidecidable set that is undecidable and strictly easier than Halting–but it has no purpose other than this; it is totally artificial.

So it is tempting to ask whether there are natural intermediate problems. Something like roots of Wurzelbrunft's polynomials.

Again: Not a single example of such a natural intermediate problem is known today. All natural semidecidable problems have ultimately turned out to be either decidable, or complete, sometimes requiring huge effort.

Formalization

There ain't much.

Computability theory is not one of the areas that has attracted much attention form the theorem proving community.

Maybe it's not sexy enough, or maybe the endless combinatorial details that are "obvious" from an intuitive perspective turn into a nightmare when one tries to nail down every little detail.

Digression: Personal Opinions

Mathematics is often presented as this pure, ethereal, crystalline structure where everything is utterly perfect and eternal. And, of course, all mathematicians agree on everything.

Nothing could be further from the truth. There are quite a few people who think that classical recursion theory has gone off the deep end, and needs a serious reset.

Theoretical computer science and complexity theory might just be the cure. Initially, the flow of ideas was all from mathematics to TCS, but there are now examples in the opposite direction.

H. Poincaré

Formerly, when one invented a new function, it was to further some practical purpose; today one invents them in order to make incorrect the reasoning of our fathers, and nothing more will ever be accomplished by these inventions.

Martin Davis

. . . but one can be quite precise in stating that no one has produced an intermediate r.e. degree about which it can be said that it is the degree of a decision problem that had been previously studied and named.

Dana Scott

70 years of research on Turing degrees has shown the structure to be extremely complicated. In other words, the hierarchy of oracles is worse than any political system. No one oracle is all powerful.

Suppose some quantum genius gave you an oracle as a black box. No finite amount of observation would tell you what it does and why it is non-recursive. Hence, there would be no way to write an algorithm to solve an understandable problem you couldn't solve before! Interpretation of oracular statements is a very fine art–as they found out at Delphi.

John Myhill

The heavy symbolism used in the theory of recursive functions has perhaps succeeded in alienating some mathematicians from this field, and also in making mathematicians who are in this field too embroiled in the details of their notation to form as clear an overall picture of their work as is desirable. In particular the study of degrees of recursive unsolvability by Kleene, Post, and their successors has suffered greatly from this defect, . . .

Hao Wang

The study of degrees 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.

Computability, Math, Physics

So far, the comments are all due to mathematicians, which is only fair since computability theory is a branch of mathematical logic.

But there is another group out there that has strong opinions about computability: a bunch of physicists.

David Deutsch

The theory of computation has traditionally been studied almost entirely in the abstract, as a topic in pure mathematics. This is to miss the point of it. Computers are physical objects, and computations are physical processes. What computers can or cannot compute is determined by the laws of physics alone, and not by pure mathematics.

Rolf Landauer

Information is not a disembodied abstract entity; it is always tied to a physical representation. It is represented by an engraving on a stone tablet, a spin, a charge, a hole in a punched card, a mark on paper, or some other equivalent. This ties the handling of information to all the possibilities and restrictions of our real physical world, its laws of physics and its storehouse of available parts.

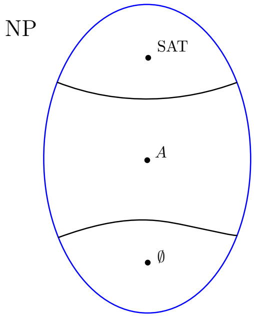
Stephen Wolfram

Principle of Computational Equivalence in 2002:

> *. . . all processes, whether they are produced by human effort or occur spontaneously in nature, can be viewed as computations.*
>
> *. . . almost all processes that are not obviously simple can be viewed as computations of equivalent sophistication.*

Clearly false in the setting of classical computability theory. But if one works in a suitable physics-like model of computation involving observers, it might actually work out.

Scaling Down



Could this picture be accurate: $\mathbb{P}$ is separate from $\mathbb{NP}$, and there are problems in $\mathbb{NP} - \mathbb{P}$ that fail to be complete.

Intermediate in $\mathbb{NP}$

Alas, here we don't know that the classes are distinct (and some notable researchers like Levin and Knuth suggest that they well may be the same). So let's assume $\mathbb{P} \neq \mathbb{NP}$ just to get off the ground.

In a rather compelling way, the situation is similar to that in classical recursion theory: if a natural problem is found to be in $\mathbb{NP}$, after some amount of effort, it will turn out that either

- the problem has a polynomial time algorithm, or
- the problem is $\mathbb{NP}$-hard.

Again, this is a matter of experience, not a theorem.

Cliff-Hangers

In his seminal 1972 paper, Karp already identified 3 potentially intermediate problems for $\mathbb{NP}$:

- Graph Isomorphism
- NonPrime (aka Composite)
- Linear Inequalities

As we now know, the last two are in fact in $\mathbb{P}$.

The first one is still a mess and seems to require massive amounts of group theory.

Ladner's Theorem, 1974

Theorem ($\mathbb{P} \neq \mathbb{NP}$)

There are intermediate problems in $\mathbb{NP}$.

The proof is similar in spirit to the Friedberg/Mučnik construction and uses requirements. As in CRT, it produces an entirely artificial example of an intermediate problem.

Since separating $\mathbb{P}$ from $\mathbb{NP}$ is probably rather difficult, it currently looks quite hopeless to find natural examples.

Battleplan

Assume a repetitive enumeration (M_e) of all deterministic polynomial time Turing machines as usual, say, clocked at $|x|^e + e$.

Also assume that we are given an enumeration (f_e) of all polynomial time computable functions.

We want to construct a set A in $\mathbb{NP}$ subject to the following constraints:

(R_e)	$A \neq \mathcal{L}(M_e)$	insure $A \notin \mathbb{P}$
(S_e)	for some φ: $\mathsf{SAT}(\varphi) \neq A(f_e(\varphi))$	insure SAT $\not\leq_m^p A$

Watering Down SAT

We are going to thin out SAT by imposing an additional condition on yes-instances, using a mystery function g:

$$A = \{ \varphi \in \mathsf{SAT} \mid g(|\varphi|) \text{ is even} \}$$

The magic function g is arithmetical and, on the face of it, makes no sense. Note that if g is polynomial time computable, then A is in $\mathbb{NP}$: we can guess-and-verify that φ is satisfiable, and then check that $g(|\varphi|)$ is indeed even (this part requires no nondeterminism).

This assumes some fixed encoding of Boolean formulae as binary strings. There are lots of possibilities, lets say we write the formula in prefix form and write variables as `x01101` (x ultimately needs to be coded in binary).

Construction

As one might expect, there is no nice explicit definition on g, we have to define the function in stages. Initially set $g(0) = g(1) = 2$.

Defining $g(m{+}1)$:

If $(\log m)^{g(m)} \geq m$ we bail out: $g(m+1) = g(m)$.

$\log m$ is the length of the input to g, so we are trying to avoid some polynomial increase in the length to be too large.

We cannot get stuck in this mode: the LHS increases logarithmically, but the RHS increases linearly; we must ultimately reach a sufficiently large m such that the condition fails and we take further action.

Construction II

Otherwise consider the following two cases, depending on parity (so we can work on both kinds of requirements).

Search for a formula φ such that $|\varphi| \leq \log m$ such that

$$\begin{array}{rl} g(m) = 2e & M_e(\varphi) \neq A(\varphi) \\ g(m) = 2e{+}1 & \mathsf{SAT}(\varphi) \neq A(f_e(\varphi)) \end{array}$$

Define the next value of g as follows:

$$g(m+1) = \begin{cases} g(m) + 1 & \text{if } \varphi \text{ exists} \\ g(m) & \text{otherwise.} \end{cases}$$

Observations

- g is non-decreasing: $g(m+1) = g(m) + 0/1$.
- If g is unbounded, then all the requirements are satisfied.
- If g is bounded, it must be ultimately constant.

So if g is unbounded, we are done: we must have found the formula φ as in the construction infinitely often. Also, we change the parity of $g(m)$ at every increase, so all the requirements must be satisfied. Thus, we have constructed an intermediate set in $\mathbb{NP}$.

Bounded Case

Case 1: g is ultimately even.

Then A is SAT minus finitely many instances.

But A is also polynomial time from the even case of the construction, contradicting our assumption $\mathbb{P} \neq \mathbb{NP}$.

Case 2: g is ultimately odd.

Then A is finite.

From the odd case of the construction, SAT reduces to A. Again, $\mathbb{P} = \mathbb{NP}$, contradicting our assumption.

Recap

In CRT, we can prove that there are semidecidable sets $\emptyset <_T A <_T \emptyset'$. There are many natural examples at level $\emptyset$ and level $\emptyset'$, but all known intermediate ones are utterly artificial.

Assuming $\mathbb{P} \neq \mathbb{NP}$, we can construct intermediate problems in $\mathbb{NP}$. And again, these are not natural the way polynomial time and $\mathbb{NP}$-complete problems are.

This may just be the nature of the beast.

Thinning Out SAT

Ladner's theorem can be interpreted as saying that if we remove a sufficiently large (but not too large) part of SAT, we are left with an intermediate set, assuming $\mathbb{P} \neq \mathbb{NP}$.

One might wonder if there are other ways to cut down on complexity.

Definition

A language $L \subseteq \Sigma^\star$ is sparse if there is some polynomial p such that $|L \cap \Sigma^n| \leq p(n)$.

In general we only have an exponential bound $|L \cap \Sigma^n| \leq |\Sigma|^n$.

Examples

Example

$a^\star b^\star a^\star$ is sparse.

Example

VC is not sparse.

Example

All tally languages $L \subseteq \{a\}^\star$ are sparse.

Example

Figure out when a regular language is sparse (depending on, say, the minimal DFA).

Exceptions

Sparse sets are interesting in that we could use them to weaken performance guarantees.

Say, we have a pretty good algorithm $\mathcal{A}$ for SAT. It would be nice if the sets of inputs where $\mathcal{A}$ messes up (correctness or running time) were sparse:

$$\{\, x \mid \mathcal{A}(x) \text{ is slow} \,\}$$

$$\{\, x \mid \mathcal{A}(x) \text{ is wrong} \,\}$$

Alas, that seems unlikely: sparse sets are not that complicated.

Ordering Truth Assignments

Think of strings in $\mathbf{2}^{\leq m}$ as partial truth assignments to m Boolean variables.

$\mathbf{2}^{\leq m}$ is the complete binary tree of depth m, and we will say that σ is to the left of τ, in symbols $\sigma \preceq \tau$, if τ is a prefix of σ, or σ lies in a branch to the left of τ.

For example, $011100 \preceq 0111$ and $011011 \preceq 0111$.

Also $0^m \preceq \tau \preceq 1^m$ for all $\tau \in \mathbf{2}^m$.

Mahaney's Theorem, 1982

Theorem ($\mathbb{P} \neq \mathbb{NP}$)

There are no sparse $\mathbb{NP}$-complete languages.

Proof.

Consider the following version of SAT. Assume φ is a Boolean formula with m variables.

$$\mathsf{SAT}_0 = \{\, \varphi\#\tau \mid \tau \in \mathbf{2}^{\leq m}, \exists\, \sigma \in \mathbf{2}^m \,(\sigma \preceq \tau \wedge \sigma(\varphi) = 1)\,\}$$

We will call the partial assignment τ a hint: we are looking for a satisfying truth assignment to the left of τ.

Note that $\varphi \in \mathsf{SAT} \iff \varphi\#1^m \in \mathsf{SAT}_0$, so this version is also hard.

But intuitively, it might be easier to check $\varphi\#\tau$ than performing a direct satisfiability test: there are fewer assignments involved.

This is certainly true for $\tau = 0^m$.

Now assume there is a sparse set S so that $\mathsf{SAT}_0 \leq^p_m S$ via a map f.

Since f is polynomial time, there is some polynomial p such that $|f(x)| \leq p(|x|)$. On inputs of size n, the reduction can only reach at most $q(n)$ points in S where

$$q(n) \geq |S \cap \Sigma^{\leq p(n)}|$$

But S is sparse, so q is another polynomial.

We are going to construct a polynomial time test for SAT_0 by constructing a polynomially small tree of hints $\tau \in \mathbf{2}^{\leq m}$.

Initially, $W = \{\varepsilon\}$.

We proceed in m rounds; in each round we extend our current hints by one more bit. So in the end we will have constructed a few truth assignments.

Informally, in each round, we start with all possible extensions $W \cdot \mathbf{2}$. We then prune the hints until the cardinality drops down to $q(n)$.

The critical part is that pruning must not destroy suitable hints, if they exist at all.

```
set $W = \{\varepsilon\}$

foreach $k = 1, \ldots, m$ do

    set $W = W \cdot \mathbf{2}$

    if $\tau_1 \prec \tau_2 \in W$ and $f(\varphi\#\tau_1) = f(\varphi\#\tau_2)$
    then delete $\tau_2$

    while $|W| > q(n)$
        delete $\prec$-minimal hint

check all remaining hints
```

Note that this computation is polynomial time by brute force: at any point during the execution, $|W| \leq 2q(n)$.

We have to make sure we are not deleting all hints that are a prefix of a satisfying assignment (if any).

If $w_1 \prec w_2 \in W$ and $f(\varphi\#w_1) = f(\varphi\#w_2)$, deleting w_2 cannot destroy a critical hint: w_1 must also be a prefix of a satisfying assignment.

If $|W| > q(n)$, then at least one hint in W maps to $\overline{S}$ under f. Hence at least one prefix does not extend to a satisfying assignment. But then the least one does not, either.

But then checking $W \subseteq \mathbf{2}^m$ provides the correct answer: if there is a satisfying assignment, then the minimal one will survive pruning. If not, our checks all come out false.

$\square$