UCT

Intermediate Problems

Klaus Sutner

Carnegie Mellon University

Spring 2022

Where Are We?

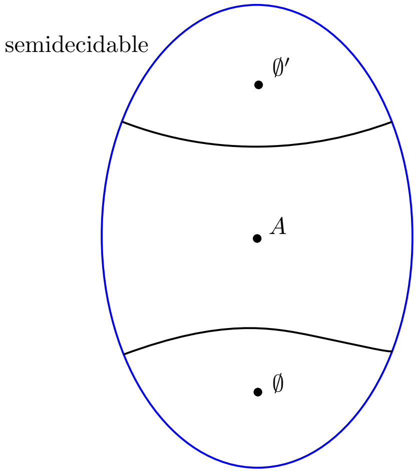
- We have a collection of practical $\mathbb{NP}$-complete problems.
- It is perfectly reasonable (though not uncontroversial) to assume that these problems do not admit polynomial time solutions.
- Suppose that is the case. What can we say about the area between $\mathbb{NP}$-complete and $\mathbb{P}$?

Public Service Announcement

The “proofs” in this lecture are very, very sketchy, the reason being that an actual careful 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 set A such that $\emptyset <_T A <_T \emptyset'$?

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$^{\text{TM}}$ either belong to the degree of $\emptyset$ or the degree of $\emptyset'$.

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.

Natural?

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

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

> In the late 19th century, famous number theorist 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 a root is intermediate.

This has never happened, not in any area of mathematics where undecidability results are known.

Friedberg/Muchnik 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. Muchnik in Russia (they published their results in 1957 and 1956, respectively).

Theorem (Friedberg, Muchnik 1956/7)

There are intermediate semidecidable problems.

What is perhaps even more surprising, Friedberg and Muchnik 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.

We'll sketch the argument below, emphasizing the main idea. There are many technical details that we will casually ignore.

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

Density

Friedberg/Muchnik 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 $A <_T B$. There 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.

How to Construct a Semidecidable Set

To describe the construction it is best to use the characterization of semidecidable sets as being recursively enumerable.

The construction proceeds in stages $\sigma \in \mathbb{N}$.

At stage σ one builds a finite set $A_\sigma \subseteq \mathbb{N}$. In the end we set

$$A := \bigcup_{\sigma \geq 0} A_\sigma$$

More precisely, there is a primitive recursive function C, the construction, such that

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

where $A_{<\sigma} = \bigcup_{\tau<\sigma} A_\tau$.

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 $C(\sigma, A_{<\sigma})$ 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 set up a very sophisticated construction that produces the set in stages, making sure that all the conditions are ultimately satisfied. In the limit, everything is fine.

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.

Of course, this is **not** a condition that is decidable, we will have to make do with finite approximations.

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.

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

Requirements

The two sets are constructed in parallel in stages. 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: in the end we need all of them to be satisfied.

- 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

$$\{e\}_\sigma^{A_{<\sigma}}(a) = b$$

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 $42 \in A_{<\sigma}$? 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 on output b we're good. Otherwise we consider the equation false.

The Catch

Suppose at the time of the query $42 \in A_{<\sigma}$ 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):

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

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):

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

The next move is practically forced: throw a into A_σ, and we have broken the equality.

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

Requirement (R_e) just received attention.

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

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

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

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

And, of course, if we fix (R'_i) 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'_i: we would like to place b into B, but we can't.

To give R'_i 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.

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/Muchnik 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.

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.

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.

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.

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.

Principle of Computational Equivalence

Stephen Wolfram 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 physics-like model of computation, it might actually work out.

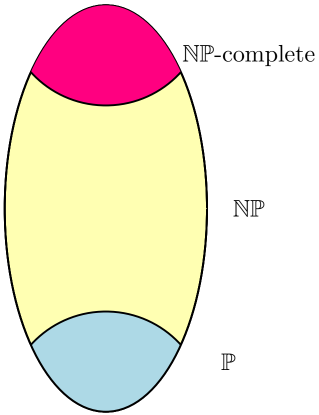
More Details

If you want to read more about the details of this priority argument, take a look at

Post's Problem

There are lots of messy details, but in the end it's just a plain induction argument.

Scaling Down



Pretty, but is this picture accurate? Is the yellow area empty?

Intermediate in $\mathbb{NP}$

Here we don't even know that the classes are distinct (and some notable researchers like Levin and Knuth claim they are not).

In 1972 Karp identified 3 potentially intermediate problems for $\mathbb{NP}$:

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

The Primality Shock

Pratt has shown that Prime is in $\mathbb{NP}$ (a beautiful application of number theory). So both Prime and NonPrime are in $\mathbb{NP} \cap$ co-$\mathbb{NP}$, and thus unlikely to be $\mathbb{NP}$-complete. Sure enough ...

Theorem (Agrawal-Kayal-Saxena 2002)

Prime is in polynomial time.

Annoyingly, their algorithm uses no more than high-school algebra.

AKS does not seem to provide a computationally superior method, probabilistic algorithms are far better.

Linear Inequalities

In Karp's version one has to decide if there exists a rational vector $\boldsymbol{x}$ such that

$$M \cdot \boldsymbol{x} \geq \boldsymbol{a}$$

where M and $\boldsymbol{a}$ are integral.

This comes down to linear programming.

Theorem (Khachiyan 1979)

Linear Programming is in polynomial time.

The proof is quite difficult and does not directly yield superior algorithms (superior to the standard simplex algorithm).

Ladner's Theorem

Theorem (Ladner 1975)

If $\mathbb{P} \neq \mathbb{NP}$, then there are intermediate problems wrto polynomial time reducibility.

The proof is quite similar to the Friedberg/Muchnik construction and 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 an enumeration (M_e) of all deterministic Turing machines, say, clocked at $|x|^e + e$.

Also assume that we are given a list (f_e) of all polynomial time computable functions (ignore arity issues).

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 $\not\leq_m^p A$

Waterin Down SAT

We are going to thin out SAT by imposing an additional condition on yes-instances:

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

Note that if the magic function 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.

As one might expect, there is no nice explicit definition on g, we have to define the function by induction (essentially the standard CRT trick of using stages).

Initially set $g(0) = g(1) = 2$.

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

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

Otherwise, consider the following two cases, depending on parity. Search for a formula φ such that $|\varphi| \leq \log m$ such that

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

If such a φ exists, set $g(m+1) = g(m) + 1$; otherwise let $g(m+1) = g(m)$.

Observations

- g is non-decreasing.
- If g is ultimately constant, then A is SAT minus finitely many instances.
  - If g is ultimately even, then A is polynomial time.
  - If g is ultimately odd, then SAT $\leq_m^p A$.
- If g is unbounded, then all the requirements are satisfied.

Done

- g ultimately even.

  A is polynomial time, but polynomial time equivalent to SAT, so $\mathbb{NP} = \mathbb{P}$, contradicting our assumption.

- g ultimately odd.

  Then A is finite, so again $\mathbb{NP} = \mathbb{P}$, contradicting our assumption.

So What?

In CRT, we can **prove** that there are semidecidable sets $\emptyset <_T A <_T \emptyset'$. Examples at level $\emptyset$ and level $\emptyset'$ are natural, but the intermediate ones are utterly artificial.

For $\mathbb{P}$ and $\mathbb{NP}$, if we assume separation, there are also intermediate problems. We have perfectly natural examples at level $\mathbb{P}$ and at level $\mathbb{NP}$, but the intermediate ones currently are nowhere near as natural.

Without the CRT result, this might sound deeply suspicious. One might even suspect that $\mathbb{P} = \mathbb{NP}$. But given the CRT mess, 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; as always, 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 $|L \cap \Sigma^n| \leq p(n)$.

Of course, 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 A for SAT. It would be nice if the sets of inputs where A messes up (correctness or running time) were sparse:

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

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

Alas, that's unlikely.

Mahaney's Theorem

Theorem

If $\mathbb{P} \neq \mathbb{NP}$, there are no sparse $\mathbb{NP}$-complete languages.

Proof.

Consider the following version of SAT. Throughout, assume $|\varphi| = n$ and write $\preceq$ for lexicographic order on $\mathbf{2}^n$.

$$\mathsf{SAT}_0 = \{\, \varphi\#v \mid v \in \mathbf{2}^{\leq k}, \exists\, \sigma \preceq v1^{n-k}\, (\sigma(\varphi) = 1)\,\}$$

We will call v a hint: look for a satisfying truth assignment "below" v.

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

But intuitively, it might be easier to check $\varphi\#v$ than performing a direct satisfiability test.

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

Let $|f(x)| \leq p(|x|)$ for some polynomial p and set

$$q(m) = |S \cap \Sigma^{\leq p(m)}|.$$

We are going to construct a polynomial test for SAT_0 by constructing a "small" tree of hints $v \in \mathbf{2}^\star$.

Initially, $W = \{\varepsilon\}$ and extend W from $\mathbf{2}^{k-1}$ to $\mathbf{2}^k$, $k \leq n$, as follows.

Informally, we start with $W \cdot \mathbf{2}$ and prune until the cardinality drops to $q(n)$. The pruning will not destroy suitable hints (if they exist).

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

**foreach** $k = 1, \ldots, n$
    **set** $W' = W \cdot \mathbf{2}$

    **if** $w_1 \prec w_2 \in W'$, $f(\varphi\#w_1) = f(\varphi\#w_2)$
    **then** delete $w_2$

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

**check** all remaining hints
```

Note that this is polynomial time by brute force: $|W'| \leq 2q(n)$.

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

If $w_1 \prec w_2 \in W'$, $f(\varphi \# w_1) = f(\varphi \# w_2)$, 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 $W \subseteq \mathbf{2}^n$ provides the correct answer (consider the minimal satisfying assignment).

$\square$