CDM

Oracles and Reductions

Klaus Sutner

Carnegie Mellon University

Where Are We?

We have seen (some of) the fundamental results in computability theory, including the dreaded recursion theorem.

We also have encountered three complexity classes: decidable, semidecidable and co-semidecidable. As one might suspect, this is just the tip of the iceberg.

The next step is to try to get a better understanding of the complexity landscape.

Halting

Our foothold in the world of undecidability is the Halting set

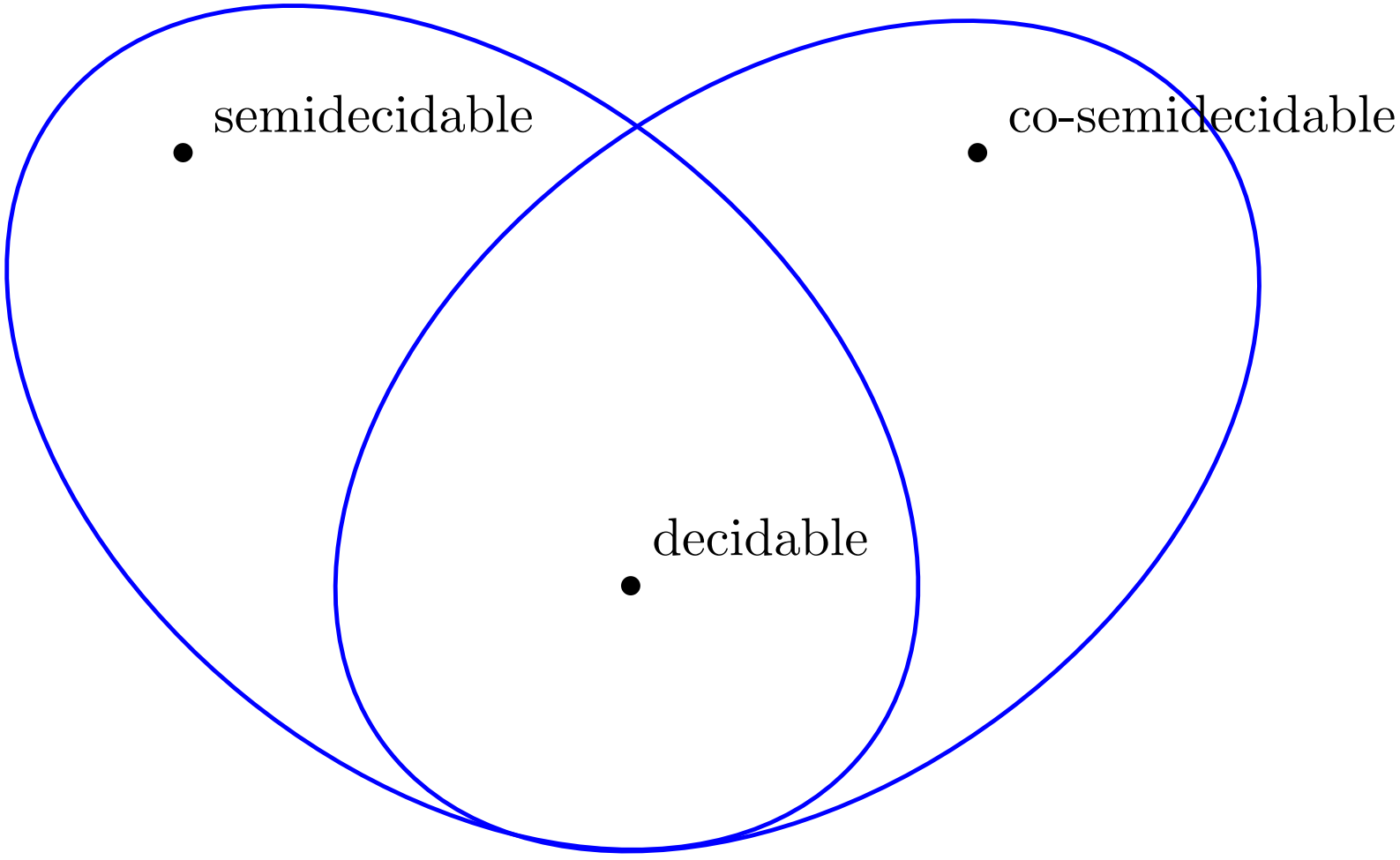
$$K = \{\, e \in \mathbb{N} \mid \{e\}(e) \downarrow \}$$

K immediately produces three types of problems:

- decidable
- semidecidable
- co-semidecidable

Is there more?

In Pictures . . .



Bad Answer

As usual, we can resort to set theory and counting arguments: there are $2^{\aleph_0}$ subsets of $\mathbb{N}$.

Only countably many of them are decidable, semidecidable and co-semidecidable.

So almost all sets $A \subseteq \mathbb{N}$ must be more complicated.

True, but fairly useless. We want concrete examples, and perhaps a nice hierarchy.

Classifying Undecidability

What we would like is some notion of reducibility: problem $B \subseteq \mathbb{N}$ is easier than problem $A \subseteq \mathbb{N}$ if an algorithm for A could be translated into an algorithm for B.

Dire Warning:

This does not mean that A is decidable, it just establishes a relationship between the two problems.

In fact, in a sense this idea is only interesting when A fails to be decidable. Right?

Turing's Wild Idea

> Let us suppose we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle as it were . . . this oracle cannot be a machine. With the help of the oracle we could form a new kind of machine (call them o-machines), having as one of its fundamental processes that of solving a given number-theoretic problem.

In Pictures . . .

Is $17 \in A$?

Yes.

Quoi?

- Oracle machines were introduced by Turing in a 1939 paper under the name of o-machines (as opposed to the original a-machines). Curiously, he never really exploited this idea (Emil Post did).

- As we will see, OTMs provide a powerful way to classify problems according to their complexity.

- Think of the oracle as a data base, created by an alien super-civilization a trillion years ago. The oracle Turing machine has access to all their wisdom.

Less Informally

Fix some set $A \subseteq \mathbb{N}$. We want to add knowledge about A to a Turing machine: the machine writes $x \in \mathbb{N}$ in a special area on the tape and then enters a magical query state q.

At this point, a genie takes over and checks whether $x \in A$. If so, the machine state is changed to q_Y, otherwise it is changed to q_N.

Then the normal computation resumes.

Definition

A Turing machine with this added facility is a oracle Turing machine (OTM) with oracle A.

For Real

The last slide is rather vague. But one can make the notion of an oracle Turing machine precise, the same way as an ordinary Turing machine can be defined precisely (in the usual pseudo-set-theory setting).

Exercise

Give a precise definition of oracle Turing machines by redefining the next-step relation.

We write M_e^A for the eth OTM with oracle A and $\{e\}^A$ for the corresponding partial function.

The Turing o-machine is the single most important concept in computability, theoretical or practical.

Comments

- The whole point behind Turing machines is that they are physically realizable, at least in principle. They capture everything that is physically possible, and more.
- But o-machines are **NOT**, though some esteemed members of the hyper-computation community fail to realize that.
- The one exception is when the oracle A is decidable: in this case we can replace magic by another Turing machine.

Decidable Oracles

From the perspective of general computability, decidable oracles are useless: instead of asking the oracle, we can simply run a corresponding computation.

But this idea is very useful in actual algorithms: suppose, for example, you have an oracle for SAT: the oracle decides whether a Boolean formula is satisfiable.

Then we can use this oracle to construct a satisfying assignment.

This can help to organize the logic of a real algorithm.

The Key Generalization

We can now generalize all the basic notions we have relative to an arbitrary oracle A: we copy our old definitions, replacing TM by OTM everywhere.

- A-computable
- A-decidable
- A-semidecidable

So computable is $\emptyset$-computable and so forth.

More Precisely . . .

Definition

Let $A, B \subseteq \mathbb{N}$. A partial function $f : \mathbb{N}^k \nrightarrow \mathbb{N}$ is computable in A or A-computable if there is an oracle Turing machine that computes f using A as an oracle.
A set B is Turing-reducible to A or A-decidable if its characteristic function is A-computable.
A set B is semidecidable in A or A-semidecidable if its semi-characteristic function is A-computable.

Turing reducibility is usually written

$$B \leq_T A$$

A Meta-Theorem

As it turns out, the machinery we developed for computable functions carries over to A-computable functions, just about verbatim.

In particular we still have a Kleene style enumeration

$$\{e\}^A$$

of all A-computable functions.

Then $B \leq_T A$ means

$$B(x) \simeq \{e\}^A(x)$$

for some index e (we abuse notation slightly by writing B for the characteristic function of B).

Claim

Every semidecidable set is K-decidable.

Proof. Here is an overly careful argument.

Let W be semidecidable. Define the following function:

$$g(x,z) \simeq \begin{cases} 0 & \text{if } x \in W, \\ \uparrow & \text{otherwise.} \end{cases}$$

Clearly, g is computable and has some index $\widehat{g}$. Then

$$g(x,z) \simeq \{\widehat{g}\}(x,z) \simeq \{S_1^1(\widehat{g},x)\}(z)$$

by the S-m-n theorem.

The map $f(x) := S^1_1(\widehat{g}, x)$ is primitive recursive.

But then $x \in W \iff f(x) \in K$, done. $\square$

Usually this would be abbreviated to something a bit more cryptic like this:

Let

$$\{f(x)\}(z) \simeq \begin{cases} 0 & \text{if } x \in W, \\ \uparrow & \text{otherwise.} \end{cases}$$

Then f is primitive recursive, blahblahblah, done.

Note that the critical point here is that an index for the last function is easily computable. One can prove this formally by applying the S-m-n theorem, but usually no one bothers.

Since an oracle TM can compute f, it can check membership in W by making just one call to the oracle and returning the same answer true or false (a bit like tail recursion).

This is a very special case, in general multiple calls are needed and the OTM has to do more work than just computing a primitive recursive function.

More on this special type of reduction (many-one reduction) later.

A Preorder

Proposition

Turing reducibility is a preorder (reflexive and transitive):

- $A \leq_T A$.
- $A \leq_T B$ *and* $B \leq_T C$ *implies* $A \leq_T C$.

Proof. Every call to oracle B in the algorithm for A can be replaced by a computation which uses calls to oracle C.

The actual computation can be absorbed by the algorithm, so that all that remains are the calls to oracle C.

□

Of course, $\leq_T$ is not symmetric. There even are incomparable semidecidable sets, but that's more complicated.

So we can lump together problems that are mutually reducible.

Definition

Two sets A and B are Turing equivalent if

$$A \leq_T B \text{ and } B \leq_T A$$

This defines an equivalence relation $\equiv_T$ whose equivalence classes are called Turing degrees.

Notation:

$$\deg_T(A) = \{ B \mid A \equiv_T B \}$$

Degrees of Unsolvability

Historically Turing degrees are also called degrees of unsolvability since they measure the distance between a problem and solvability.

The only degree that is easy to describe is

$$\deg_T(\emptyset) = \text{all decidable sets.}$$

How about the degree of the Halting Problem, $\deg_T(K)$?

Note that by definition Turing reducibility can handle complements:

$$\overline{A} \leq_T A$$

So we cannot conclude that $A \leq_T K$ implies that A is semidecidable.

So $\deg_T(K)$ contains all semidecidable sets and all co-semidecidable sets. Unfortunately, there is even more, but it's a bit difficult to describe these additional sets; we'll skip.

Just Two Results

Theorem (Friedberg, Muchnik 1955/57)

There exist two semidecidable sets that are not comparable wrto Turing reducibility.

Theorem (Sacks 1964)

Let A and B semidecidable such that $A <_T B$. Then there exists a semidecidable C such that $A <_T C <_T B$.

How Many Degrees

Note that $\deg_T(A)$ is always a countable collection, no matter what $A \subseteq \mathbb{N}$ is.

It follows from standard general abstract nonsense ($\aleph_0 < 2^{\aleph_0}$) that there must be uncountably many Turing degrees.

OK, but we live in the computational universe and we would much prefer more concrete examples: we would like to pin down some degrees that have clear computational meaning, other than just $\deg_T(\emptyset)$ or $\deg_T(K)$.

A Convergence Question

Here is a failed attempt. Consider the following question, an apparently harder variant of the Halting problem:

Does $\{e\}$ converge on some input?

Claim

This problem is semidecidable but not decidable.

Proof.

We need to search for some x such that $\{e\}(x)\downarrow$.

Note that we cannot simply start by computing $\{e\}(0)$, then $\{e\}(1)$ and so on.

Dovetailing

Instead, we organize a big parallel computation, organized in stages $\sigma \geq 0$.

Stage σ:

Compute $\{e\}_\sigma(x)$ for all $x < \sigma$.

If one of them terminates, halt.

Otherwise, go to stage σ^+.

This method is used in many places and referred to as dovetailing. It is quite similar to the standard argument that $\mathbb{N} \times \mathbb{N}$ is countable.

Hardness

To show that convergence is not decidable, note that there is a primitive recursive function f such that

$$\{f(e)\}(z) \simeq \begin{cases} 0 & \text{if } \{e\}(e)\downarrow \\ \uparrow & \text{otherwise.} \end{cases}$$

But then $e \in K$ iff $\{f(e)\}$ converges on some input.

Or, more explicitly, writing K_2 for the set of Yes-instances of our convergence problem, we have

$$e \in K \iff f(e) \in K_2$$

Since f is computable, K_2 cannot be decidable. $\square$

Same Old

Note that we could be lazy and simply use Rice's theorem: we are asking whether $W_e \neq \emptyset$.

But then $W_e = W_{e'}$ clearly implies $e \in K_2 \iff e' \in K_2$.

Also, $\emptyset \neq K_2 \neq \mathbb{N}$. Done.

But our proof shows a bit more: $K \leq_T K_2$. Since $K_2 \leq_T K$ we have $\deg_T(K) = \deg_T(K_2)$.

A Small Hierarchy

Here is a better attempt. Consider the following classes of r.e. sets:

$$\begin{aligned}
\mathsf{FIN} &= \{\, e \in \mathbb{N} \mid W_e \text{ is finite} \,\} \\
\mathsf{INF} &= \{\, e \in \mathbb{N} \mid W_e \text{ is infinite} \,\} \\
\mathsf{TOT} &= \{\, e \in \mathbb{N} \mid W_e = \mathbb{N} \,\} \\
\mathsf{REC} &= \{\, e \in \mathbb{N} \mid W_e \text{ is decidable } \,\}
\end{aligned}$$

By Rice's theorem, they are all undecidable. But we would like to understand their Turing degrees.

FIN $\equiv_T$ INF is clear.

How Bad Can It Be?

Intuitively, FIN is not computable since

$$e \in \mathsf{FIN} \iff \exists\, b \forall\, x, \sigma\, (x \in W_{e,\sigma} \Rightarrow x < b)$$

But REC is even worse:

$$\begin{aligned} e \in \mathsf{REC} &\iff \exists\, e'\, (W_e = \mathbb{N} - W_{e'}) \\ &\iff \exists\, e' \forall\, x\, (\forall\, \sigma\, (W_{e,\sigma} \cap W_{e',\sigma} = \emptyset) \wedge \exists\, \tau\, (x \in W_{e,\tau} \cup W_{e',\sigma})) \end{aligned}$$

Theorem

$$\emptyset <_T K <_T \mathsf{FIN} <_T \mathsf{REC}$$

$$\mathsf{FIN} \equiv_T \mathsf{INF} \equiv_T \mathsf{TOT}$$

One Proof

We will show that

$$K \leq_T \text{INF}$$

To see this, note that there is a primitive recursive function f such that

$$\{f(e)\}(z) \simeq \begin{cases} 0 & \text{if } e \in K, \\ \uparrow & \text{otherwise.} \end{cases}$$

Then $e \in K \iff f(e) \in \text{INF}$, done.

Another Proof

We will show that

$$\mathsf{INF} \leq_T \mathsf{TOT}$$

As a warmup exercise, consider the downward closure operation

$$\mathsf{dc}(A) = \{\, x \in \mathbb{N} \mid \exists\, a \in A\, (a \geq x) \,\}$$

So $\mathsf{dc}(\{123, 10^{10}, 100!\}) = \{0, 1, \ldots, 100!\})$ and $\mathsf{dc}(\mathrm{Primes}) = \mathbb{N}$.

In general

$$\mathsf{dc}(A) = \begin{cases} \{0, \ldots, \max A\} & \text{if } A \text{ is finite,} \\ \mathbb{N} & \text{otherwise.} \end{cases}$$

Effectivizing the Argument

So dc(A) translates from infinite to total, fairly close to what we want.

But note that dc(W) is r.e. whenever W is r.e. Moreover, we can compute an index for dc(W) from an index for W. In fact, there is a primitive recursive function f that does it:

$$\{f(e)\}(z) \simeq \begin{cases} 0 & \text{if } \exists\, u\, (u \geq z \wedge u \in W_e), \\ \uparrow & \text{otherwise.} \end{cases}$$

Then W_e is infinite iff $W_{f(e)} = \mathbb{N}$.

Hence we have the desired reduction: $e \in \mathsf{INF} \iff f(e) \in \mathsf{TOT}$, done.

Recall: Oracles

We can attach an orcale to a Turing machine without changing the basic theory.

$\{e\}^A$	eth function computable with oracle A
$W_e^A = \mathrm{dom}\{e\}^A$	eth r.e. set with oracle A

The constructions are verbatim the same. For example, a universal Turing machine turns into a universal Turing machine plus oracle.

The Use Principle

We continue to confuse a set $A \subseteq \mathbb{N}$ with its characteristic function. For any n write $A \upharpoonright n$ for the following finite approximation to the characteristic function:

$$(A \upharpoonright n)(z) \simeq \begin{cases} A(z) & \text{if } z < n, \\ \uparrow & \text{otherwise.} \end{cases}$$

For any such function α write $\alpha \sqsubset A$ if $\alpha = A \upharpoonright n$ for some n. Then

$$\{e\}^A(x) = y \iff \exists\, \alpha \sqsubset A \left(\{e\}^\alpha(x) = y\right)$$

with the understanding that the computation on the right never asks the oracle any questions outside of its domain.

Of course, a divergent computation may use the oracle infinitely often.

Generalized Halting

Definition

Let $A \subseteq \mathbb{N}$. The (Turing) jump of A is defined as

$$A' = K^A = \{\, e \mid \{e\}^A(e) \downarrow \}$$

So $\emptyset' = K^\emptyset = K$ is just the ordinary Halting set.

The nth jump $A^{(n)}$ is obtained by iterating the jump n times.

Lemma

A' is A-semidecidable but not A-decidable.

Proof.

The proof is verbatim the same as for the ordinary Halting problem.

□

So we get an infinite hierarchy of more and more complicated problems:

$$\emptyset,\ \emptyset',\ \emptyset'',\ \emptyset''',\ \ldots,\ \emptyset^{(n)},\ldots$$

In reality, though, nothing much beyond $\emptyset^{(4)}$ seems to play a role (except for problems that lie entirely outside of this hierarchy).

No End in Sight

- Even worse, we can collect all these sets into a single one, essentially by taking a disjoint union:

$$\emptyset^\omega = \{ \langle e, n \rangle \mid e \in \emptyset^{(n)} \}$$

- And nothing stops us from forming $(\emptyset^\omega)'$, $(\emptyset^\omega)''$ and so on. We can iterate the jump transfinitely often: ω^2 times, ω^ω times, $\varepsilon_0 = \omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}$ times.

- Headache, anyone?

Jump Properties

Let $A, B \subseteq \mathbb{N}$.

Theorem

- *A' is r.e. in A.*
- *A' is not Turing reducible to A.*
- *B is r.e. in A iff $B \leq_m A'$.*

Proof.

The first two parts are verbatim re-runs of the oracle-free argument.

Proof, Cont'd

For part (3) suppose B is r.e. in A. Hence there is a primitive recursive function f such that

$$\{f(x)\}^A(z) \simeq \begin{cases} 0 & \text{if } x \in B, \\ \uparrow & \text{otherwise.} \end{cases}$$

This function is A-computable since we can replace B on the right by W_e^A.

But then $x \in B \iff f(x) \in A'$, done.

Proof, Cont'd

Lastly suppose $B \leq_m A'$, say, $x \in B \iff f(x) \in A'$.

To enumerate B given A as oracle, proceed in stages

Stage s:

Compute $f(x)$ for all $x = 0, 1, \ldots, s-1$.

Enumerate A'_s (use oracle A for this).

If any of the $f(x)$ appear in A'_s, enumerate the corresponding x's into B.

Theorem

A is Turing equivalent to B iff A' is one-one-equivalent to B'.

Proof.

Assume $A \leq_T B$.

There is a primitive recursive function f such that

$$\{f(e)\}^B(z) \simeq \begin{cases} 0 & \text{if } \{e\}^A(z)\downarrow, \\ \uparrow & \text{otherwise.} \end{cases}$$

But then $e \in A' \iff f(e) \in B'$.

We can force f to be injective by doing something useless like counting to e first.

Proof, Cont'd

Now assume $A' \leq_1 B'$, say, $e \in A' \iff f(e) \in B'$.

There are a primitive recursive functions g_1, g_2 such that

$$\{g_1(e)\}^A(z) \simeq \begin{cases} 0 & \text{if } e \in A, \\ \uparrow & \text{otherwise.} \end{cases}$$

$$\{g_2(e)\}^A(z) \simeq \begin{cases} 0 & \text{if } e \notin A, \\ \uparrow & \text{otherwise.} \end{cases}$$

Now we combine f and the g_i to show how a TM with oracle B can determine membership in A.

Proof, Cont'd

$$
\begin{aligned}
e \in A &\iff g_1(e) \in A' \\
&\iff f(g_1(e)) \in B' \\
&\iff \{f(g_1(e))\}^B(f(g_1(e))) \downarrow \\
&\iff \exists\, \beta \sqsubset B\, \{f(g_1(e))\}^\beta(f(g_1(e))) \downarrow
\end{aligned}
$$

Likewise

$$
e \notin A \iff \exists\, \beta \sqsubset B\, \{f(g_2(e))\}^\beta(f(g_2(e))) \downarrow
$$

Since one of the two computations must converge we can decide which with oracle B.

The Jump Hierarchy

One writes $\emptyset^{(n)}$ for the Turing degree of of $\emptyset^{(n)}$. Then

- $\emptyset$ is the degree of all decidable sets.
- $\emptyset'$ is the degree of K.
- $\emptyset''$ is the degree of FIN and TOT.
- $\emptyset'''$ is the degree of REC.

A Horror

Nothing stops us from constructing

$$\emptyset^\omega = \{ \langle n, x \rangle \mid x \in \emptyset^{(n)} \}$$

This is essentially the ω-iterate of the jump on $\emptyset$. Note that

$$\emptyset <_T \emptyset' <_T \emptyset'' <_T \ldots <_T \emptyset^{(n)} <_T \ldots <_T \emptyset^\omega$$

And, of course, we could apply the jump to $\emptyset^\omega$.

We won't go there.

Jump Theorem

One might wonder which degrees are of the form $\boldsymbol{a}'$.

Theorem (Friedberg)

Let $\boldsymbol{a} \geq_T \emptyset'$. Then $\boldsymbol{a} = \boldsymbol{b}'$ for some $\boldsymbol{b}$.

In other words, the range of the jump operator is the cone $\boldsymbol{x} \geq_T \emptyset'$.

Better Reductions

The fact that Turing degrees are closed under complements wrecks semidecidability. In a sense, this means that Turing reducibility is too coarse for semidecidable sets.

It would be nice to have some other notion of reduction $\preceq$ such that

$$B \preceq A, A \text{ semidecidable} \quad \text{implies} \quad B \text{ semidecidable}$$

Finding the right notion of reducibility is absolutely critical for lower complexity classes like $\mathbb{NP}$.

Many-One

Here is a type of reduction that is suggested by some of the examples above.

Definition

Let $A, B \subseteq \mathbb{N}$. B is many-one reducible to A if there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that

$$x \in B \iff f(x) \in A.$$

If function f is in addition injective then B is one-one reducible to A.

In symbols: $B \leq_m A$ and $B \leq_1 A$.

Nothing New

In other words: we can only ask a single question of the oracle, and whatever answer the oracle returns is also our answer.

We have already seen several examples of this kind of reduction, they seem to be quite natural.

One-one reductions in addition are required to ask different questions to the oracle for different inputs.

This is actually a little bit weird, there is no good algorithmic reason why two instances x and x' should not produce the same queries $f(x) = f(x')$.

Of course, set-theoretically this makes perfect sense. And, it turns out to produce some nice results.

Downward Closure

Proposition

$$A \leq_1 B \quad \textit{implies} \quad A \leq_m B \quad \textit{implies} \quad A \leq_T B$$

The opposite implications are all false, but it requires a bit of effort to separate many-one and one-one reductions.

Proposition

- *$A \leq_m B$ and B decidable implies that A is decidable.*
- *$A \leq_m B$ and B semidecidable implies that A is semidecidable.*

Our old proofs have already quietly used many-one reductions.

Lemma

All semidecidable sets are many-one reducible to K.

Lemma

$K \leq_m$ INF *and* $K \leq_m$ TOT

Many-One and One-One Degrees

As before with Turing reductions one can collect mutually reducible sets into a degree and obtains equivalence relations.

The equivalence classes are correspondingly called many-one degrees and one-one degrees.

In symbols:

$$A \equiv_m B \qquad \text{and} \qquad A \equiv_1 B$$

So every many-one degree is contained in a Turing degree but not the other way around, we get a finer partition with these limited types of reductions.

Definition

A set $C \subseteq \mathbb{N}$ is many-one hard (one-one hard, Turing hard) for the class of semidecidable sets if for all A r.e.: $A \leq_m C$ (or $A \leq_1 C$, $A \leq_T C$).

C is many-one complete (one-one complete, Turing complete) for the class of semidecidable sets sets if C is r.e. and C is many-one hard (one-one hard, Turing hard).

It is not difficult to fabricate a hard set: just take the disjoint union over the whole class. But completeness can be difficult.

So we know that K is many-one complete for the class of semidecidable sets.

Future Attractions

In general, one considers a class $\mathcal{C} \subseteq \mathfrak{P}(\mathbb{N})$ of sets.

Given a reducibility $\preceq$ (a preorder on $\mathfrak{P}(\mathbb{N})$) we can define $(\mathcal{C}, \preceq)$-hardness: A is hard if

$$\forall\, X \in \mathcal{C}\, (X \preceq A)$$

$(\mathcal{C}, \preceq)$-completeness means: $(\mathcal{C}, \preceq)$-hardness plus membership in $\mathcal{C}$.

We will use this approach over and over in complexity theory.

Lemma

The Halting set K is one-one complete (for semidecidable sets).

Proof.

Suppose A is r.e. We already know that there is a primitive recursive function f such that

$$\{f(x)\}(z) = \begin{cases} 0 & \text{if } x \in A, \\ \uparrow & \text{otherwise.} \end{cases}$$

So $x \in A \iff f(x) \in K$.

f will be injective in any reasonable environment. If not, we can insist that $\{f(x)\}$ first counts to x before it starts the actual computation.

□

Variants

One can define variants of K that are easily seen to lie in the same one-one degree – and are thus really the same as K from the point of view of information content.

$$K_0 = \{ \langle e, x \rangle \mid x \in W_e \}$$
$$K_1 = \{ e \mid W_e \neq \emptyset \}$$

Here $\langle e, x \rangle$ is understood to be a standard coding function.

Note that K_0 is trivially one-one hard for r.e.: we have coded all r.e. sets into a single one. The important fact is that K_0 is itself still r.e. (since there are universal machines).

Proposition

K, K_0 and K_1 are all one-one equivalent.

Proof.

Use the same trick as in the last lemma.

□

Recursive Permutations

Suppose $f : \mathbb{N} \to \mathbb{N}$ is a computable permutation. For example, f could interchange $2x$ and $2x + 1$ (yes, yes, boring).

Clearly $f(A)$ has the same complexity as A: we can use f to translate back and forth. This idea is captured in the next definition.

Definition

Two sets $A, B \subseteq \mathbb{N}$ are recursively isomorphic if there is a recursive permutation p of $\mathbb{N}$ such that $p(A) = B$.
In symbols: $A \equiv B$.

For example, all infinite and co-infinite decidable sets are recursively isomorphic (e.g., $42\mathbb{N} + 17$ and the primes).

Myhill's Isomorphism theorem

There is an analogue to the Schröder-Bernstein-Cantor theorem (Dedekind actually had a much better proof) that associates the existence of computable injections in both directions with the existence of a computable bijection.

Theorem (Myhill 1955)

$$A \equiv B \iff A \equiv_1 B.$$

In other words, the one-one degrees are simply obtained by applying a recursive permutation to the given set.

Thus K, K_0 and K_1 can all be obtained from each other by recursive permutations of $\mathbb{N}$.

Proof Sketch

Suppose $A \leq_1 B$ via f and $B \leq_1 A$ via g.

We may safely assume that A and B are both infinite.

We define a computable permutation h in stages using a zig-zag construction: $h = \bigcup h_\sigma$ where h_σ is finite and h_σ is computable uniformly in σ.

Stage $\sigma = 0$: $h_0 = \emptyset$

Stage $\sigma > 0$, even:

Assume that $H = h_{<\sigma}$ is injective and
$\forall\, x \in \mathrm{dom}\, H\, (x \in A \iff H(x) \in B)$.

We make sure that h is defined on argument $x = \sigma/2$.

If already $H(x)\downarrow$, do nothing and go to the next stage.

Proof Cont'd.

Otherwise compute the zig-zags

$$f(x), f \circ H^{-1} \circ f(x), f \circ (H^{-1} \circ f)^2(x), \ldots$$

As f and H are injective, there can be no repetitions in this sequence.

Hence for some i: $y = f \circ (H^{-1} \circ f)^i(x) \notin \operatorname{rng} H$.

Set $h_\sigma(x) = y$.

Stage $\sigma > 0$, odd:

Entirely similar, replace f, $h_{<\sigma}$ by g, $h_{<\sigma}^{-1}$.

□