How computability relates to logic

Professor Cian Dorr

13th December 2022

New York University

Review: three key definitions

Definition

Partial function f (from n-tuples of strings to strings) is [Py-]**computable** iff there is a Py-program A such that whenever $t = f(s_1, \ldots, s_n)$, $t = [\![A]\!]^{[\texttt{input1} \mapsto s_1], \ldots, [\texttt{inputn} \mapsto s_n]}(\texttt{result})$.

Definition

Where R is a relation on strings, R is [Py-]**decidable** iff the function that maps every n-tuple in R to `Y` and every other n-tuple of strings to `N` is computable.

Definition

Where R is a relation on strings, R is [Py-]**semi-decidable** iff some partial function that maps every n-tuple in R to `Y` and does not map any other n-tuple to `Y` is computable.

For the rest of the day I'll drop the prefix 'Py-'.

An alternative characterization

Alternative characterization of semi-decidability

R is semi-decidable iff there is some computable partial function f whose domain of definition is R.

Proof, left-to-right: take the program A_R that halts with `result` set to `Y` when run with input in R, and add the line `while result != "Y":` at the end (followed by a newline, 4 spaces, and another newline). Right-to-left: take the program A_f that computes f, and tack on the line `result = "Y"` at the end.

Combining computable functions

Since we can string Py-programs together to make new Py-programs, we can combine computable partial functions to make other computable partial functions. For example:

Simple Fact

If partial functions f and g are computable, $g \circ f$ is computable.

Proof: consider the program A_f ($\leftarrow$ program that computes f)

```
input = result
```

A_g ($\leftarrow$ program that computes g)

Analogous facts apply to partial functions of multiple arguments. E.g.

- If f, g, h are computable partial functions of 2 arguments, $[\langle x, y\rangle \mapsto f(g(x,y), h(x,y))]$ is also computable.
- If f is a computabl partial function of 2 arguments, $[x \mapsto f(x,x)]$ and $[\langle x, y\rangle \mapsto f(y,x)]$ are computable too..

Combining decidable sets and relations

Simple Fact

If R and S are decidable, $R \cup S$, $R \cap S$, $R \setminus S$, and R^{-1} (for binary R) are too.

Proof of first three claims: consider

```
$A_R$
if result == "N":
    $A_S$
```

```
$A_R$
if result == "Y":
    $A_S$
```

```
$A_S$
if result == "N":
    $A_R$
else:
    result = "N"
```

Slightly less simple fact

If R and S are semi-decidable, $R \cup S$, $R \cap S$, and R^{-1} are too.

(This is trickier to show for $\cup$.) NB: Even if R and S are semidecidable, $R \setminus S$ needn't be!

Simple observations

Blatantly obvious fact

If a partial function is computable, its domain of definition is semidecidable.

Proof: the Py-program that computes f also semi-decides its domain of definition.

Application: we saw last week that we can compute the partial function that takes a Py-program A and the string-encoding g' of an assignment function g and returns the string-encoding of $[\![A]\!]^g$ if A halts on input g and is otherwise undefined.

- So, $\{\langle A, g' \rangle \mid A$ is a Py-program that halts on decode$(g')\}$ is semi-decidable.
- So is $\{\langle A, s \rangle \mid A$ is a Py-program that halts on $[\texttt{input} \mapsto s]\}$.
- So is $\{A \mid A$ is a Py-program that halts on $[\texttt{input} \mapsto A]\}$.
- So is the set of all Py-programs that halt on the empty input.

Some simple observations about these notions

Simple fact

Every computable partial function is semidecidable.

Proof: consider A_f

```
if result == input2:
    result = "Y"
else:
    result = "N"
```

Note that if f is total this will halt on all inputs, so we have:

Corollary

Any computable **total** function is decidable.

Actually, the converse of that Simple Fact holds too:

Non-obvious fact which I may not have time to prove

Any semi-decidable partial function is computable.

A set that isn't even semidecidable

We can think of any Py-program A as semi-deciding the set of strings s such that it halts on the input s (i.e., $[\![A]\!]^{[\texttt{input} \mapsto s]}$ is defined); for every semidecidable set, there's a program that semidecides it in this sense.

There are only countably many Py-programs and uncountably many sets of strings, so by Cantor's theorem, not all the sets are semidecidable.

And when we remember how Cantor's theorem is proved, we can give an example: the set of all programs that do not halt when given themselves as input (i.e., for which $[\![A]\!]^{[\texttt{input} \mapsto A]}$ is undefined). There can't be a program that halts exactly when it is given such a program as input, since then it halts when given itself as input iff it doesn't halt when given itself as input.

An undecidable but semi-decidable set

Given this, the set H of all programs that **do** halt when given themselves as input can't be decidable.

For the set of *all* Py-programs is decidable. (Writing a Py-program that checks whether its input is a well-formed Py-program is tedious but feasible.)

So if H was decidable, the set of Py-programs *not* in H would be decidable too (since the complement of a decidable set is decidable). But it isn't decidable, since it isn't even semi-decidable. We thus have an example of a set that's semi-decidable but not decidable.

- Another example: the relation $\{\langle A, s\rangle \mid A$ halts on $[$`input` $\mapsto s\}$.
- A famous example: the set of programs that halt on empty input. (If we had a program to decide this, we could build a program that, when run on $[$`input1` $\mapsto A$, `input2` $\mapsto s]$ sets `input` to a new program that first sets `input` to be `s` and then runs A, and then tests whether *that* program halts on empty input.)

Projections

Definition

If R is an $n+1$-ary relation and S is an n-ary relation, S is the **projection** of R iff for all $x_1, \ldots, x_n$, $Sx_1 \ldots x_n$ iff there exists y such that $Rx_1 \ldots x_n y$.

Simple Fact

If a relation is decidable, its projection is semi-decidable.

Proof: given inputs $x_1, \ldots, x_n$, loop through all strings in order of length, and for each one y, run the program that tests whether $Rx_1 \ldots x_n y$. If this program outputs `Y`, halt and output `Y`. Otherwise, keep looping.

In fact we can strengthen this fact to a biconditional.

Handy Fact

A relation is semi-decidable iff it is the projection of some decidable relation.

Sketch of proof: modify last week's "interpretation" program to take one extra input, a number n (represented by a string of `*`s); decrement n by one with each step through the loop, and stop if it reaches zero. This shows that for any program A, the relation *A halts with* `result` *set to* `Y` *on input s in at most n steps* is decidable. But if A semi-decides X, X is the projection of this relation.

Using this, we can prove the previously stated

Non-obvious fact

Every semi-decidable partial function is computable.

Proof: if 1-ary function f is semi-decidable, it's the projection of some decidable 3-ary R. To compute $f(s)$, loop through all pairs $\langle t_1, t_2 \rangle$ of strings (in order of total length) and test whether Rst_1t_2; if the answer is ever `Y`, stop and output t_1.

An application in logic

It's easy to see that the set of all *proofs* (for a given decidable signature) is decidable: just go through the lines in reverse order and for each one, check for each of the 18 rules whether it follows from preceding lines by that rule, by looping through all the preceding lines. Hence if Ax is decidable, the relation that holds between Y and P when Y is a proof of P from Ax is decidable. Thus,

Important fact that explains why we are talking this in the first place

Any decidably axiomatizable theory is semi-decidable.

Corollary: semi-representable relations are semi-decidable

Any relation semi-represented in a decidably axiomatizable theory is semi-decidable.

Proof: where $P(x_1, \ldots, x_n)$ is the semi-representing formula, we can write a program that gets the standard labels of all its inputs, substitutes them into P, and then semi-decides whether the resulting sentence is in the theory.

Why semi-decidability is so-called

Important fact

Suppose X and Y are two **non-overlapping** semi-decidable sets, and $X \cup Y$ is **decidable**. Then X and Y are both decidable.

Special case: if a set and its complement are both semi-decidable, they are both decidable.

Intuition: to decide whether a string belongs to X, first decide if it's in $X \cup Y$. If it is, start two programs running simultaneously, one of which semi-decides membership in X and one of which semi-decides membership in Y. If the first one halts, stop the second one and output `True`; if the second one halts, stop the first one and output `False`.

We can implement this in Py by tweaking our 'interpreter' program from last week to run two programs simultaneously (keeping track of a separate assignment for each one), stopping if either of them stop.

Two applications to logic

Representable Relations are Decidable

Any relation represented in a consistent, decidably axiomatizable theory is decidable.

Proof: if R is represented, its complement is too; since the theory is consistent, both are semi-represented; so both are semi-decidable; so both are decidable.

Crucial Fact That Sheds New Light On The Incompleteness Theorem

Any decidably axiomatizable, negation-complete theory is decidable.

Proof: suppose T is decidably axiomatizable and negation-complete. Then T is semi-decidable, and so is $T' := \{P \mid \neg P \in T\}$. If T is inconsistent, it's just the set "Sent" of all sentences of the signature, which is decidable. (We assume the signature itself is decidable.) Otherwise, since T is negation-complete, $T \cup T' = \text{Sent}$, which is decidable, so both T and T' are decidable by the Important Fact.

Computability and logic

A key link

Theorem: Representability of Computable Functions

Every computable function is capturable in Min.

Given our Representability Theorem we can derive this from the following lemma

Σ_1-Definability of Computable Functions

Every computable partial function is Σ_1-definable in the standard string structure.

To establish *this*, we can appeal to the following result we had about Σ_1-definability and closures: if set X and relations $R_1 \ldots R_n$ are all definable by bounded formulae, then the closure of X under $R_1 \ldots R_n$ is definable by a Σ_1 formula.

In the present instance, we are interested in the set of triples $\langle A, g, h\rangle$ where A is a Py-program and $h = [\![A]\!]^g$. It was defined as the closure of

$$X \coloneqq \{\langle [], g, g\rangle \mid g \text{ is a Py-assignment}\}$$

under the relations

$$R_1 \coloneqq \{\langle\langle A, g', h\rangle, \langle B, g, h\rangle\rangle \mid B = v\,\texttt{ = }\,t \oplus A \text{ and } g' = g[v \mapsto [\![t]\!]^g]\}$$
$$R_2 \coloneqq \{\langle\langle A, g, h\rangle, \langle B, g, h\rangle\rangle \mid B = \texttt{ while }\,t_1\texttt{!=}t_2\texttt{:} \oplus \texttt{ }\,C \oplus A \text{ and } [\![t_1]\!]^g = [\![t_2]\!]^g]\}$$
$$R_3 \coloneqq \{\langle\langle B, g', h\rangle, \langle C, g, g'\rangle, \langle B, g, h\rangle\rangle \mid B = \texttt{while }\,t_1\texttt{!=}t_2\texttt{:} \oplus \texttt{ }\,C \oplus A \text{ and } [\![t_1]\!]^g \neq [\![t_2]\!]^g]\}$$

Of course we can't talk directly about such triples in the language of strings. But we can code the triples as single strings (by coding the two assignments as strings, then combining the three strings using an appropriate separator). Once this is done, it's straightforward to show that X, R_1, R_2, R_3 are all definable by bounded formulae.

Corollary

A relation is decidable iff it is representable in Min.

Proof: For the left-to-right direction, suppose function symbol f (in a definitional extension of Min) captures the function that maps all members of X to `Y` and all other strings to `N`. Then $\text{Min} \vDash f(\langle s \rangle) = "Y"$ when $s \in X$ and $\text{Min} \vDash f(\langle s \rangle) \neq "Y"$ when $s \notin X$, so the formula $f(x) = "Y"$ represents X. We already proved the right-to-left direction.

Corollary

A relation is semi-decidable iff it is semi-representable in Min.

Proof: if n-ary relation R is semi-decidable, it's the projection of some $n+1$-ary decidable S. When $P(x_1, \ldots, x_{n+1})$ represents S, $\exists x_{n+1}\, P(x_1, \ldots, x_{n+1})$ semi-represents R.

The Essential Undecidability Theorem

Recall

Tarksi's Undefinability Theorem

No consistent theory that extends Min represents itself.

Given the result we just established, we can infer the

Essential Undecidability Theorem

No consistent theory that extends (or interprets) Min is decidable.

- The extension to theories that merely *interpret* Min follows since any interpretation is a computable function from one language to another, and the preimage of a decidable set under a computable function is decidable.

A striking consequence of Essential Undecidability

As a corollary of the Essential Undecidability Theorem, we have

Church's Undecidability Theorem

The set of logical truths in the language of strings is undecidable.

Proof: where M is the conjunction of the axioms of Min (which is finitely axiomatizable), P is a theorem of Min iff $M \to P$ is a logical truth, so any decision procedure for logical truth could be turned into a decision procedure for Min.

A new route to a familiar result

Now we can return to that:

Crucial Fact That Sheds New Light On The Incompleteness Theorem

Any theory that is decidably axiomatizable and negation-complete is decidable.

This fact and the Essential Undecidability Theorem combine to give us a new, and illuminating, proof of

Gödel's First Incompleteness Theorem

No consistent theory that extends (or interprets) Min is decidably axiomatizable and negation-complete.