Computation Theory
Supplementary notes on decidability

Andrej Ivašković (ai294)
Compiled on: 23rd August 2019

These notes are supplementary to the main course materials. The main reason for writing them is exploring some concepts in more depth and exploring some key ideas in reduction proofs.

I first give a slightly more formal treatment of *reductions*. You will see them in more detail in *Complexity Theory*, but some practice with this approach should make you more comfortable with undecidability proofs. I go on to discuss how we approach undecidability proofs of 'practical' problems, especially the ones that deal with analysing program flow and behaviour (which come back many times in *Optimising Compilers*). Finally, I give an intuitive statement of Rice's theorem, which is a simple and useful result that can instantly give you proofs of undecidability in some cases.[1]

You should be acquainted with the notions of Minsky's register machines (RM), RM computability and decidability, enumerating RMs, universal RM and a proof of the undecidability of the Halting Problem.

1 Decidability and reductions

Traditionally, textbooks on Computation Theory have a 'grammar-centric' view: Turing machines are introduced as an extension of push-down automata, which are, in turn, extensions of deterministic finite automata. This then results in a natural hierarchy of languages 'recognised' by each kind of automaton.[2] This course takes a different view: computable partial functions over accepting a language. However, since we regularly encounter the problem of determining whether a set is decidable, it is useful to consider both approaches.

Let us remind ourselves of the definition of decidability given in this course:

[1] Sadly, using Rice's theorem without proof is not something I recommend in an exam environment.

[2] This is the *Chomsky hierarchy* – if you haven't seen it in *Discrete Mathematics*, you will see it in *Formal Models of Language* and *Complexity Theory*. DFAs recognise regular languages, PDAs recognise context-free languages, and Turing machines recognise *recursively enumerable languages*. There are also *context-sensitive* languages between the recursively enumerable and context-free ones.

Definition 1.1

A set $S \subseteq \mathbb{N}$ is *register machine decidable* if the total function $\chi_S : \mathbb{N} \to \{0, 1\}$, defined with:

$$\chi_S(x) = \begin{cases} 1, & x \in S \\ 0, & x \notin S \end{cases}$$

is register machine computable.

In other words, we see 0 and 1 as `REJECT` and `ACCEPT` answers.

From now on, I will mostly use the terms 'decidable' and 'computable', without explicitly mentioning the formal model of computation (the ones covered in the course are all equivalent).

One of the most common tasks you do in this course is proving undecidability of a particular set. The most significant such problem is the Halting Problem. You go on to use *reductions* – these, informally, we tend to use to create arguments of the form 'if you could show that S is decidable, then the set T would also be decidable'.

Starting with the Halting Problem:

Theorem 1.1

The set $\{\langle e, x\rangle \mid \varphi_e(x)\downarrow\}$ is register machine undecidable.

Proof. Seen in lectures. □

Definition 1.2

Given two sets S_1 and S_2, a *reduction* of S_1 to S_2 is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in S_1$:

$$x \in S_1 \iff f(x) \in S_2$$

The existence of a reduction from S_1 to S_2 tells us that, in some sense, decidability of S_2 is 'at least as hard' as the decidability of S_1.

Theorem 1.2

If $f : S_1 \to S_2$ is a reduction of S_1 to S_2 and S_1 is undecidable, then S_2 is also undecidable.

Proof. This is the contrapositive of Exercise 5 in the exercise sheet. □

The direction here seems somewhat counter-intuitive at first. We use this result when we want to show that S_2 is undecidable, and S_1 is usually the set seen in the Halting Problem – that is, $\{\langle e, x\rangle \mid \varphi_e(x)\downarrow\}$. Essentially, we show that any instance of the problem of deciding membership of S_1 can be somehow 'encoded' as an instance of the problem of deciding membership of S_2, where f is this 'encoding'.

Let us now see how a simple example fits into this framework.

Example 1.1. Show that $\{e \mid \varphi_e(0)\downarrow\}$ is undecidable.

Let $S = \{e \mid \varphi_e(0)\downarrow\}$, and let $H = \{\langle e, x\rangle \mid \varphi_e(x)\downarrow\}$. We want to find a reduction of H to S. Just doing a projection $\pi_1 : \langle e, x\rangle \mapsto e$ will not work, since this is not a reduction: a program might never terminate for input 0, but it might terminate for some other input. The reduction f must map every $\langle e, x\rangle$ into a program with code $f(e)$ that essentially does the same thing as e, but sets its input to x. Such a program transformation is not difficult:

$$f(\langle e, x\rangle) = \lceil \text{set registers to values specified by } x\text{; append machine } e\rceil$$

This is a 'simple enough' operation, and it is computable (in fact, it can be written explicitly using the list processing operations used in the universal register machine). f is a reduction of H to S because:

- for every instance $\langle e, x\rangle \in H$, the machine with code $f(\langle e, x\rangle)$ will halt when started with input 0;
- for every $f(\langle e, x\rangle) \in S$, we know that the machine with this code halts, and it can only terminate when e halts when started with x.

Since H is undecidable, then S must also be undecidable. □

2 Operational behaviour of programs

Not all program analysis is undecidable, but it usually is. In this section I discuss the "sometimes it can be decidable" kinds of problems, which are mainly concerned with the program execution, less so about the result of the computation.

Example 2.1. Let

$$S = \{\langle e, x\rangle \mid \text{RM } e \text{ halts in at most 100 steps when started with input x}\}$$

Even though it looks like the Halting Problem, this turns out to be decidable! This is merely due to the fact that there is a constant upper bound here (which could be arbitrarily large).

Recall the universal register machine U: it takes a RM code e and an input x, then simulates the execution of e when started with x and returns the output. We can introduce small modifications to U to solve this problem. Let us introduce a special `N` register that keeps track of how many steps the machine has taken so

far. This counter is incremented every time an instruction of e is executed. If the value of this counter exceeds 100, halt the machine and output 0. Otherwise, if e halts before then, halt and output 1. This machine decides S. □

It is usually the case that, whenever these kinds of sets are decidable, we somehow make use of universal register machines.

Example 2.2. Let:

$$S = \{e \mid \text{RM } e \text{ loops at least once when started with all registers zeroed}\}$$

Looping at least once is not a property that is conveniently stated in terms of φ_e. Furthermore, this is a decidable set: the property here says nothing about the program never terminating, merely that it only looped at least once.

A program loops at least once if its trace (list of all program counter values seen during its execution) has duplicates. Once again we look at the universal register machine U. Let us modify it somewhat, and construct a similar machine V.

- There is no input x – set all of the registers to 0 initially.
- V maintains in a special register `S` the list (that is, its natural number representation) of all program points seen so far – initially $\langle\rangle$.
- Before V considers the instruction in e with label $\mathtt{L}_k$, it first checks whether `S` contains k as its member: if it does, it halts with the value 1; if it does not, it adds k as the head of `S` and continues.
- If the execution of e halts, then V halts with the return value 0.

All of this is computable, and V will always terminate (the register machine e has a finite number of instructions). Hence S is decidable. □

However, universal register machines are not a universal solution, there remain undecidable problems. Sometimes they are very small variations of decidable ones.

Example 2.3. Let:

$$S = \{\langle e, x, \ell\rangle \mid \text{RM } e \text{ started with } x \text{ eventually reaches the label } \ell\}$$

In other words, this is like answering the question "will the program trace contain the instruction with label ℓ". This is intuitively undecidable – let us prove that it is indeed so.

We construct a reduction of the Halting Problem. Consider an arbitrary RM e and input x. The RM e can be transformed so that all `HALT`s are replaced with a single one, say the label with the least index ℓ, and all of the 'transitions' are updated appropriately. All 'invalid transitions' can become valid by making them point to the `HALT` instruction at L_ℓ. This yields a computable function f that maps $\langle e, x\rangle$ to $\langle e', x, \ell\rangle$, where e' is the RM just described. This f is also a reduction of H to S:

- whenever e halts when started with x, this means that e' will also halt when its input is x, and the only `HALT` is at label $\mathtt{L}_\ell$;
- whenever the execution of e' when started with x reaches $\mathtt{L}_\ell$, this means that the execution of e when started with x halts.

Again, since H is undecidable and f is a reduction of H to S, we conclude that S is undecidable.

This shows that even determining whether a particular part of the code is reachable is undecidable. Note, however, that there exist conservative overapproximations that can determine some of the instructions which are surely unreachable. □

3 Rice's theorem

Rice's theorem is a simple and general statement about the decidability of certain sets. You may see it as just another proof that everything interesting turns out to be undecidable. In fact, you will sometimes see it quoted as 'anything non-trivial is undecidable', which is very misleading. Properly stating the theorem and seeing how it is applicable is part of the challenge. After that, the applications are mostly simple.

If you want to avoid using Rice's theorem for a particular problem, you can adapt the proof given here – it is a general approach you can use in many different scenarios.

3.1 Statement

Theorem 3.1

Let P be a property of codes representing register machines. Consider the set $S = \{e \mid P(\varphi_e)\}$. If both of the following are true:

- P is a property that depends solely on φ_e
- $\emptyset \neq S \neq \mathbb{N}$

then S is undecidable.

Proof. See subsection 3.3. □

Pay attention the important requirement that the property P should depend on φ_e, not e: we are not talking about the 'small step' behaviour of the machine here, but rather its 'big step' or 'functional' behaviour. We call these kinds of properties P *semantic*.

It is not uncommon to see the property P depend on e in some sources. You should not forget that it still has to be a semantic property.

3.2 Applications

Before delving into how we use it, let us first have a look at two simple cases when Rice's theorem cannot be applied.

Example 3.1. Consider the following set: $S = \{e \mid e \text{ is even}\}$. It is very clearly decidable. Rice's theorem does not apply here, as the property given here does not depend on φ_e. The same holds for $S = \{e \mid e \text{ contains an even number of labels}\}$. □

Example 3.2. Let $S = \{e \mid \varphi_e \text{ is computable}\}$. While the property does depend on φ_e, it in fact holds for all e: every φ_e is, by definition, a computable function. Since $S = \mathbb{N}$, Rice's theorem cannot be used here either. □

Rice's theorem gives us an alternative, simple method for establishing the decidability of some problems. Consider the following two examples 'sanity checks'.

Example 3.3. Let $S = \{e \mid \varphi_e \text{ is total}\}$. There are clearly computable functions that are total, say the identity function $x \mapsto x$, and those that are not, say the totally undefined partial function $\perp$ (its RM contains an infinite loop). Since the property relies on φ_e, the set S is undecidable. □

Example 3.4. Let $S = \{e \mid \varphi_e(x) = 0 \text{ for at least one input } x\}$. This is a property that depends on φ_e, so this is fine. The constant `zero` function is clearly computable and is in S, and the constant `one` function is not in S. Hence S is undecidable by Rice's theorem. □

However, Rice's theorem really shines in slightly more involved examples, where a reduction of the Halting Problem might not be immediately obvious.

Example 3.5. Consider the set:

$$T = \{\langle e, k\rangle \mid \text{whenever RM } e \text{ halts, it returns a value less than } k\}$$

Intuitively, this should be undecidable. We cannot use Rice's theorem, as T is not the 'right kind of set'. However, we can get some insight by fixing a particular value of k: if T were decidable, then we would be able to establish membership of, say, $\langle e, 7\rangle$. Thus we first solve a different problem.

For some positive natural number k, let:

$$S_k = \{e \mid \text{whenever RM } e \text{ halts, it returns a value less than } k\}$$

While this looks operational, this is actually a purely semantic property:

$$S_k = \{e \mid (\forall x \in \mathbb{N})(\varphi_e(x)\downarrow \implies \varphi_e(x) < k)\}$$

so we can apply Rice's theorem.[3] S_k is non-trivial, since $\texttt{zero} \in S_k$ and $\texttt{const}_k \notin S_k$, where $\texttt{const}_k$ is the computable total function that is constantly equal to k

[3]The notation I use here is of the form $(\forall x \in S)(Q(x))$, for a set S and predicate Q. Alternative ways of writing it are $\forall x \in S.\, Q(x)$, $(\forall x \in S)Q(x)$ and others.

(its RM simply has k $\mathtt{R}_0^+$ instructions in succession, and then a `HALT`). Thus S_k is undecidable.

Now consider the following reduction f of S_7 to T:

$$f : e \mapsto \langle e, 7 \rangle$$

It is not difficult to show f is a reduction. Thus T is undecidable by Theorem 1.2. □

3.3 Proof*

There are multiple proofs of Rice's theorem. The one I present here is just a reduction of the Halting Problem.

Proof. Let $S = \{e \mid P(\varphi_e)\}$ be non-trivial for some semantic property P.

Consider a register machine program that never halts and let c be its code.

We may, without loss of generality, assume $\neg P(\varphi_c)$ (if it does not hold, just consider the set $\mathbb{N} \setminus S$).

Since S is non-trivial, there also a program with code $b \in S$, that is, $P(\varphi_b)$ holds.

Consider the set $H = \{\langle e, x \rangle \mid \varphi_e(x)\downarrow\}$. Define the function $f : H \longrightarrow S$:

$$f(e, x) = \left\lceil \begin{array}{l} \text{save input in some register } \mathtt{Y} \text{ not used by } e; \\ \text{run } e \text{ with input } x; \\ \text{run } b \text{ with input } \mathtt{Y} \text{ and return result} \end{array} \right\rfloor$$

This function is computable, we can use universal RMs to compute it. Let us show that f is a reduction.

- Assume $\langle e, x \rangle \in H$. Since e halts when started with x, $\varphi_{f(e,x)} = \varphi_b$. Since $P(\varphi_b)$ holds, $f(e, x) \in S$.
- Assume $\langle e, x \rangle \notin H$. In this case $\varphi_{f(e,x)} = \varphi_c$, since RM $f(e, x)$ will never halt and the corresponding partial functions are totally undefined. Since $\neg P(\varphi_c)$, we conclude $f(e, x) \notin S$.

Thus S is undecidable. □

4 A note on numbering*

So far, we used the notation φ_e to represent the partial function computed by register machine with code e. For a RM M, this coding $e = \lceil M \rceil$ was the simple enumeration stated in the notes. However, we could have just as easily come up with a different enumeration scheme.

The key idea was to find a bijection f between $\mathbb{N}$ and the set of all RMs. If we had found another such bijection g, then they would really just be renaming – in fact, f and g induce another bijection $h : \mathbb{N} \to \mathbb{N}$ such that $f = g \circ h$.

One consequence of this is that all of the decidability results are independent on the numbering used. This makes sense – the Halting Problem should be undecidable whatever bijection we use. Thus, if you find yourself reading some other materials on computation theory, you will find that some of the theorems seen here contain 'for every numbering'.

These kinds of encodings of languages or structures as natural numbers are commonly referred to as *Gödel numberings*.[4] One example is based on prime factorisation of numbers: given an n-tuple $\langle x_1, x_2, \ldots, x_n \rangle$, encode it using the product of the first n primes, raised to corresponding powers:

$$\ulcorner \langle x_1, x_2, \ldots, x_n \rangle \urcorner = 2^{x_1} \cdot 3^{x_2} \cdot \ldots \cdot p_n^{x_n}$$

5 Thanks

Special thanks go to the people who read these notes and gave valuable suggestions: Dhruv Makwana, Domagoj Stolfa, Shaun Steenkamp.

6 Additional exercises

You will encounter several problems about decidability of sets during your revision. Here are some of the supplementary ones, including a few which are more easily solved using Rice's theorem.

Hopefully, with practice, you will be comfortable approaching any such problem you might encounter in an exam environment.

Exercise 6.1. Show that the set of numbers that are codes of register machines that do not contain any loops is decidable.

Exercise 6.2. Is the set

$$\{\langle e, x, i \rangle \mid \text{RM } e \text{ will not modify register } \mathtt{R}_i \text{ when started with } x\}$$

decidable?

Exercise 6.3. Use Rice's theorem to show that that the set $\{\langle e, f \rangle \mid \varphi_e = \varphi_f\}$ is undecidable.

Exercise 6.4. Say that a register machine M is *rectractable* if it takes a single register $\mathtt{R}_1$ as its input (and zeroes all the other registers before usage) and there

[4] These are named after *Kurt Gödel*, whose *incompleteness theorems* carry special significance and are closely related to early results of computation theory.

exists a register machine N such that it also takes only $\mathtt{R}_1$ as its input (with same initialisations to zero) such that, whenever M halts for a certain input, N will halt when its input is M's output, and it then returns what M's input originally was.

Show that the set of all numbers that are codes of rectractable register machines is undecidable.

Exercise 6.5. Attempt 2015 Paper 6 Question 3. It discusses a variant of the *Busy Beaver* problem – the function s in the question grows too fast to be computable.