On Rice's theorem*

Hans Hüttel

October 2001

We have seen that there exist languages that are Turing-acceptable but not Turing-decidable. An important example of such a language was the language of the Halting Problem

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a } \mathsf{TM} \text{ and } M \text{ accepts } w\}$$

which, as we discovered, was *not* Turing-decidable. We can now use this result to show that a plethora of other decision problems are also undecidable. The idea comes from Chapter 5 of ITTTOC [5], namely that of *reducibility* – we provide an algorithmic transformation faithfully transforming any instance of the our problem P which we know to be undecidable (here P is the Halting Problem) to a corresponding instance of our new problem P'.

If we had a decision algorithm for P', we would then also have a decision algorithm for our old problem P as we could simply prefix our decision algorithm for P' with our transformation algorithm. However, we also know that such an algorithm cannot exist, so P' must be undecidable.

1 Properties

In this note we shall use the notion of reducibility to establish to what extent (if any) one may decide properties of the input languages of Turing machines. The presentation in my note is inspired by that of [2].

1.1 Property checkers would come in handy

It would be nice indeed if we could determine if the input language of a Turing machine had some given property:

Given Turing machine M, does $L(M)$ have property $\mathcal{S}$?

*This note is a translation by the author of [1]

An example of a property of Turing-acceptable languages could be, say, that the language in question was regular. If we had an algorithm for determining whether the language of a Turing machine was regular we would have a tool that we could use to determine whether we might as well construct a finite-state automaton.

It would also be nice if we could determine if a Turing machine accepted all strings that contained some specific sequence of characters. Then we would have a tool for determining e.g. if all data that we assumed to be correct were indeed correctly handled.

Sadly, such property checkers do not exist. This is a consequence of Rice's theorem which states that any non-trivial property of Turing-acceptable languages is undecidable.

1.2 What is a property, mathematically speaking?

We shall now prove Rice's theorem. To do this, we need to formulate a mathematical counterpart of the everyday notion of "having a property".

A property of Turing-acceptable languages corresponds to the set of languages possessing the particular property. For instance the property "being context-free" corresponds to the class $\{L \mid L \text{ is a context-free language}\}$. Conversely, any class of Turing-acceptable languages $\mathcal{S}$ gives us the property 'being a member of $\mathcal{S}$'.

Which is why the following definition makes sense.

Definition 1.1 *Let $\mathcal{S}$ be an arbitrary class of Turing-acceptable languages sprog. $\mathcal{S}$ is called a* property*. We say that a language L* has the property $\mathcal{S}$ *if $L \in \mathcal{S}$. The property $\mathcal{S}$ is called* trivial *if it is either the empty property of the class of all Turing-acceptable languages. We define the 'property language' of $\mathcal{S}$ as $L_{\mathcal{S}} = \{\langle M \rangle \mid L(M) \in \mathcal{S}\}$, i.e. the language of all machine encodings that describe a machine recognizing a language that has the property $\mathcal{S}$.*

In what follows, whenever we speak of properties, we therefore think of classes of Turing-acceptable languages.

Example 1.1 The following are examples of non-trivial properties of Turing-acceptable languages (why?)

- The class $\{\emptyset\}$ consisting of the empty language (not to be confused with the empty property $\emptyset$!)
- The class of languages that have the empty string ϵ as a member
- The class of context-free languages

- The class of regular languages
- The class of Turing-decidable (i.e. recursive) languages

□

2 Rice's theorem

Rice's theorem was originally shown in [4].

Theorem 2.1 (Rice's theorem) *If $\mathcal{S}$ is a non-trivial property of Turing-acceptable languages, then the problem 'Does $L(M)$ have the property $\mathcal{S}$?' is undecidable.*

The proof of Rice's theorem consists of a reduction from the Halting Problem. We show how one could use a property-checking algorithm to devise an algorithm for solving the Halting Problem.
PROOF: Consider a non-trivial property $\mathcal{S}$. We show that the problem 'Does $L(M)$ have the property $\mathcal{S}$?' is undecidable by a reduction from the Halting Problem.

In the remainder of our proof we shall assume that the empty language $\emptyset$ is not a member of $\mathcal{S}$. We then construct, given a machine/input description $\langle M, w\rangle$ a Turing machine M' such that $L(M')$ has the property $\mathcal{S}$ if and only if M halts when given w as input. Our assumption that $\emptyset \notin \mathcal{S}$ gives us that this is equivalent to stating that $L(M') \neq \emptyset$ if and only if M halts when given w as input.

It is no loss of generality to assume that $\emptyset \notin \mathcal{S}$. For if we wanted to show that $\mathcal{S}$ was undecidable and $\emptyset \in \mathcal{S}$, we could simply consider the complementary property $\overline{\mathcal{S}}$. This property is undecidable if and only if $\mathcal{S}$ is.

Since $\mathcal{S}$ is a non-trivial property of Turing-acceptable languages, there must be some $L \in \mathcal{S}$. Since L is Turing-acceptable, there exists a Turing machine recognizing L. We call this machine M_L.

Now assume that $\mathcal{S}$ is decidable. This will mean that the property language $L_{\mathcal{S}}$ is Turing-decidable (recursive). This in turn means that there exists a Turing machine $M_{\mathcal{S}}$ which for any machine description $\langle M'\rangle$ will determine whether $L(M') \in \mathcal{S}$ or not.

We can now solve the Halting Problem by an algorithm presented in Figure 1.

The pre-processor **Prep** takes $\langle M, w\rangle$ as input and outputs a new machine description $\langle M'\rangle$, where $L(M') \in \mathcal{S}$ iff M accepts w (i.e. if $\langle M, w\rangle \in L_H$).

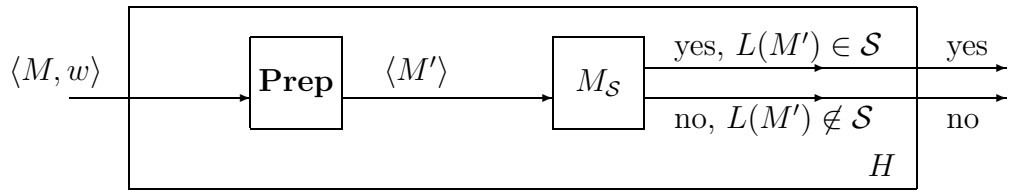

Figure 1: Solving the Halting Problem using the property checker for $\mathcal{S}$

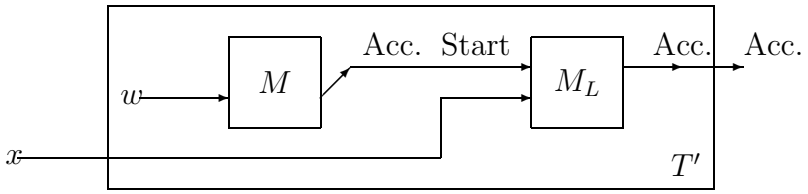

Figure 2: The machine M' constructed by **Prep**

But we must also describe the internal workings of **Prep**. In other words: What kind of M' is constructed by **Prep**? This is revealed in Figure 2.

M' starts its operation by ignoring its actual input x and simulates M using the fixed input w. If M does *not* halt when given w, M' will obviously not halt on input x no matter what x is. If, on the other hand, M *does* halt on w we start M_L with x as its input and accept x iff M_L halts on x.

In other words: If M halts when given w, M' accepts exactly the same strings as M_L, i.e. $L(M') = L$, and we know that $L \in \mathcal{S}$. If M does not halt when given w, M' will not accept any inputs, i.e. $L(M') = \emptyset$ and by assumption $\emptyset \notin \mathcal{S}$.

And now we can decide the Halting Problem using the following algorithm: For any given input string $\langle M, w \rangle$, use **Prep** to construct $\langle M' \rangle$. Next, use $M_{\mathcal{S}}$ to determine if $L(M')$ has the property $\mathcal{S}$. $L(M')$ has, as we have just seen, the property $\mathcal{S}$ iff M halts on w. But the Halting Problem is

undecidable, so $M_{\mathcal{S}}$ cannot exist. □

Stop and think 2.1 When (if ever) do we run the machine M' constructed by **Prep**? □

3 Consequences of Rice's theorem

Theorem 2.1 has a whole bunch of consequences, some of which I will present in the following.

3.1 Undecidability results concerning input languages

The following results are immediate from Rice's theorem:

Corollary 3.1 *The following problems are all undecidable. Given Turing machine M,*

1. *– does M halt on an empty input tape?*
2. *– does M halt for any inputs at all?*
3. *– does M halt for all inputs?*
4. *– is $L(M)$ regular ? Context-free ? Turing-decidable?*

PROOF: Follows from Example 1.1 □

We can also use Rice's theorem to give a somewhat different proof of Theorem 5.4 in ITTTOC [5].

Corollary 3.2 *Given Turing machines M_1 and M_2 it is undecidable if $L(M_1) = L(M_2)$.*

PROOF: Assume that the above problem were decidable and could be decided by a Turing machine $M_=$. But then we could decide the problem $L(M) = \emptyset$ by giving $M_=$ the inputs $\langle M \rangle$ and $\langle M_\emptyset \rangle$, where $M_\emptyset$ is any Turing machine that does not accept any inputs. Contradiction! □

3.2 Undecidability results about general grammars

Another corollary (or a collection of such, if you want) is about general grammars.

Corollary 3.3 *The following problems are all undecidable: Given a general grammar G over terminal alphabet Σ,:*

1. *– is it the case that $\epsilon \in L(G)$?*
2. *– is it the case that $L(G) \neq \emptyset$?*
3. *– is it the case that $L(G) = \Sigma^\star$?*
4. *– is $L(G)$ regular ? Context-free ? Turing-decidable?*

3.3 Complexity checkers are impossible, too

Rice's theorem has repercussions in complexity theory as well, as it shows that there can be no hope of finding a means of automatic complexity analysis.

Questions such as 'Does Turing machine M run in polynomial time?' are also undecidable, as the class of languages that are decidable in polynomial time is a non-trivial property. This follows from the fact that some languages are known not to be decidable by any polynomial-time Turing machine.

Actually one can show that no matter what time complexity function $T(n)$ you consider, there will be some language not recognizable by a machine with time complexity $O(T(n))$. So complexity checkers cannot exist!

3.4 Don't despair completely ...

Does Rice's theorem tell us that all questions concerning Turing machines and general grammars are undecidable? The answer to that is no. The theorem only concerns the properties of the language accepted by a machine, not the machine itself. For instance, the problem 'Does a Turing machine have more than 7 states?' is obviously decidable. Rice's theorem tells us that we cannot predict the interesting aspects of the behaviour of an algorithm by algorithmic means.

References

[1] Hüttel, H. *Om Rice's sætning*, Aalborg University 2001.

[2] Hopcroft, J.E. og J.D. Ullman. *Introduction to Automata, Languages and Computation*, Addison-Wesley 1979.

[3] Machtey, M. og P.R. Young. *An Introduction to the General Theory of Algorithms*, North-Holland, 1978.

[4] Rice, H.G. Classes of recursively enumerable sets and their decision problems, *Trans. AMS* **89**:25–59.

[5] Sipser, M. Introduction to the Theory of Computation, First edition, PWS Publishing 1997.