Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 23

Recursively enumerable languages

slides by Graham Farr

Overview

- recursively enumerable (r.e.) languages
- relationship with decidability
- enumerators
- non-r.e. languages

Decidability

Recall:

A language L is decidable if and only if there exists a Turing machine T such that

$$\begin{aligned} \text{Accept}(T) &= L \\ \text{Reject}(T) &= \overline{L} \\ \text{Loop}(T) &= \emptyset. \end{aligned}$$

Reminder: $\overline{L} = \Sigma^* \setminus L$, where Σ is the alphabet.

Recursively enumerable languages: definition

A language L is **recursively enumerable** if there exists a Turing machine T such that

$$\text{Accept}(T) = L$$

Strings outside L may be *rejected*, or may make T *loop forever*.

Recursively enumerable: synonyms

recursively enumerable (r.e.)

= computably enumerable

= partially decidable

= Turing recognisable (used in Sipser)

= type 0 (in Chomsky hierarchy)

= *computable*

. . . but risk of confusion, as "computable" is sometimes used for "decidable".

Decidable versus r.e.

Every decidable language is recursively enumerable.

Is every recursively enumerable language decidable?

Consider:

$$\text{HALT} = \{T : T \text{ halts, if input is } T\}$$

This is the language corresponding to the Halting Problem.

We know it's not decidable.

Is it recursively enumerable?

Decidable versus r.e.

Let M be a Turing machine wh[illegible] as input, a Turing machine T and

- simulates what happens w[illegible]run with *itself* as its input.
- If T stops (in any state), M[illegible]s.

Here, M could be obtained by modifying a UTM.

$$
\begin{aligned}
\text{Accept}(M) &= \text{HALT} \\
\text{Reject}(M) &= \emptyset \\
\text{Loop}(M) &= \overline{\text{HALT}}
\end{aligned}
$$

Decidable versus r.e.

So HALT is recursively enumera

So some recursively enumerable languages are not decidable.

Consider the list of undecidable languages given in Lecture 22.

Which ones are recursively enumerable?

Decidable versus r.e.

Theorem.
A language is decidable if and only if both it and its complement are r.e.

Proof.
($\Longrightarrow$)
Let L be any decidable language.

We have seen that every decidable language is r.e. So L is r.e.

Now, the complement of a decidable language is also decidable.
(See Lecture 20, comments on closure properties of the class of decidable languages.)

So $\overline{L}$ is also decidable, and therefore also r.e.

So L and $\overline{L}$ are both r.e.

Decidable versus r.e.

$(\Longleftarrow)$

Let L be any language such that both L and $\overline{L}$ are both r.e.

Since they are each r.e., there exist Turing machines M_1 and M_2 such that

$$\begin{aligned} \text{Accept}(M_1) &= L \\ \text{Accept}(M_2) &= \overline{L}. \end{aligned}$$

Note, each of these TMs might *loop forever* for inputs they don't accept.

Construct a new Turing machine M' that simulates *both* M_1 and M_2:

Input: x

Repeatedly:

Do one step of M_1. If it **accepts**, then Accept.
Do one step of M_2. If it **accepts**, then Reject.

Decidable versus r.e.

M' is a decider:

- every string belongs to eit[illegible],
- therefore is accepted by eit[illegible] M_1 or M_2,
- therefore will eventually be either accepted or rejected by M'.

Furthermore, M' accepts x if and only if M_1 accepts x.

So M' is a decider for L.

So L is decidable. $\square$

A non-r.e. language

Is *every* language recursively enumerable?

Consider:

$$\overline{\text{HALT}} = \{ \text{[illegible]ops forever, if input is } T \}$$

Assume $\overline{\text{HALT}}$ is r.e.

We already know that HALT is r.e.

So, both HALT and its complement are r.e.

Therefore, by the previous theorem, HALT is decidable.

Contradiction!

Therefore $\overline{\text{HALT}}$ is not r.e.

Enumerators

Definition

An **enumerator** is a Turing machine which outputs a sequence of strings.

This can be a finite or infinite sequence.

If it's infinite, then the enumerator will never halt.

It never accepts or rejects; it just keeps outputting strings, one after another.

- If the sequence is finite, then the enumerator may stop once it has finished outputting. But the state it enters doesn't matter.

Enumerators

Definition

A language L is **enumerated** by an enumerator M if

$$L = \{\text{all strings in the sequence outputted by } M\}$$

Members of L may be outputted in any order by M, and repetition is allowed.

Theorem

A language is recursively enumerable if and only if it is enumerated by some enumerator.

Enumerators and r.e. languages

Theorem

A language is recursively enumerable if and only if it is enumerated by some enumerator.

Proof.

$(\Longleftarrow)$

Let L be a language, and let M be an enumerator for it.

Construct a Turing machine M' as follows:

> Input: a string x
>
> Simulate M, and for each string y it generates:
>
> > Test if $x = y$. If so, accept; otherwise, continue.

A string x is accepted by M' if and only if it is in L.

So Accept$(M') = L$. So L is r.e.

Enumerators and r.e. languages

$(\implies)$ Let L be r.e. Then there is a TM M such that $\text{Accept}(M) = L$. Take all strings, in order:

$$\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}, \mathtt{aaa}, \mathtt{aab}, \mathtt{aba}, \ldots$$

Simulate the execution of M on each of these strings, in parallel.

As soon as any of them stops and accepts its string,
we pause our simulation, output that string, and then resume the simulation.

CAREFUL:

Infinitely many executions to simulate, but we only have finite time!
How do we schedule all these simulations?

Enumerators and r.e. languages

Denote the strings by $x_1, x_2, \ldots$

Algorithm:

For each $k = 1, 2, \ldots$

For each $i = 1, \ldots, k$:

Simulate the next step of the execution of M on x_i

(provided that execution hasn't already stopped).

If this makes M accept, then

output x_i and skip i in all further iterations;

else if this makes M reject, then

output nothing, and skip i in all further iterations.

Enumerators and r.e. languages

This algorithm can be impleme[illegible] Turing machine.
Any string accepted by M will [illegible] be outputted.
So this is an enumerator for L. $\square$

This result explains the term "recursively enumerable" (and "computably enumerable").

It also explains why r.e. languages are sometimes called *computable*, since there is a computer that can *compute* all its members (i.e., can generate them all).

Exercises

Theorem.
A language L is r.e. if and only if
there is a decidable two-argument predicate P such that

$$x \in L \iff \exists y : P(x, y).$$

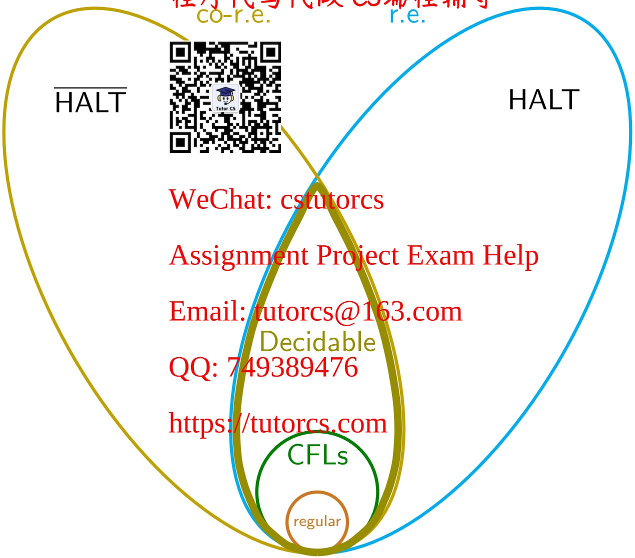
This P is a *verifier*:
if you are given y then you can use P to *verify* that x is in L (if it is).

But it may be hard to *find* such a y.

Theorem.
If $K \leq_m L$ and L is r.e. then K is r.e.

Recursively enumerable languages

Recursively enumerable languages



Revision

- definition of recursively enumerable languages
- relationship between decidability and recursive enumerability
- enumerators and their relationship with r.e. languages
- a language that is r.e. but not decidable, with proof
- a language that is not r.e., with proof

Reading: Sipser, pp. 170, 209–211.

Preparation: Sipser, pp. 275–286.