Monash University
Faculty of Information Technology

Lecture 18
Recursively enumerable languages

Slides by Graham Farr (2013).

Supplementary material by David Albrecht (2011).

FIT2014 Theory of Computation

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

- **Accept(T) = L**
- **Reject(T) = $\overline{L}$**

  (i.e., $\Sigma^* \setminus L$, where Σ is the alphabet)
- **Loop(T) = $\varnothing$**

Definition: recursively enumerable

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

- **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:

HALT = { ***T*** : ***T*** 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 which takes, as input, a Turing machine ***T*** and

- simulates what happens when ***T*** is run with *itself* as its input,
- If ***T*** stops (in any state), ***M*** accepts.

Here, *M* could be obtained by modifying a UTM.

Accept(*M*) = HALT

Reject(*M*) = ∅

Loop(*M*) = $\overline{\text{HALT}}$

Decidable versus r.e.

So HALT is recursively enumerable.

So some recursively enumerable languages are not decidable.

Consider the list of undecidable languages given in Lecture 17.

Which ones are recursively enumerable?

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 = { 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:

($\Leftarrow$)   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'*** iff it is in ***L***.

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

Enumerators and r.e. languages

(⇒) Let L be r.e. Then there is a TM M such that **Accept(M) = L**. Take all strings, in order:

ε, a, b, aa, ab, ba, bb, aaa, aab, aba, ……….

Simulate the execution of M on each of these strings, in parallel. As soon as any of them stops and accepts its string, then 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 $x_1, x_2, \ldots, x_i, \ldots$

Algorithm:

```
For each $k = 1, 2, \ldots\ldots\ldots$
    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 implemented by a Turing machine.

Any string accepted by ***M*** will eventually be outputted.

So this is an enumerator for ***L***.

Q.E.D.

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).

Is every language recursively enumerable?

A non-r.e. language

Consider:

$\overline{\text{HALT}}$ = { ***T*** : ***T*** ***loops forever***, if input is ***T*** }

If $\overline{\text{HALT}}$ is r.e., then **Accept(*M₀*)** = $\overline{\text{HALT}}$ for some $\boldsymbol{M_0}$.

We'll want a machine that either accepts or loops forever.

So, we define a new machine, ***M***, from $\boldsymbol{M_0}$ as follows.

On any input:

M behaves just like $\boldsymbol{M_0}$ until $\boldsymbol{M_0}$ stops (if it does).

If $\boldsymbol{M_0}$ accepts, then ***M*** accepts.

If $\boldsymbol{M_0}$ rejects, then ***M*** loops forever.

A non-r.e. language

Is $\boldsymbol{M}$ in $\overline{\text{HALT}}$?

$\boldsymbol{M}$ is in $\overline{\text{HALT}}$ $\Rightarrow$ $\boldsymbol{M_0}$ accepts $\boldsymbol{M}$
$\Rightarrow$ $\boldsymbol{M}$ doesn't loop forever for input $\boldsymbol{M}$

$\Rightarrow$ $\boldsymbol{M}$ is not in $\overline{\text{HALT}}$

$\boldsymbol{M}$ is not in $\overline{\text{HALT}}$ $\Rightarrow$ $\boldsymbol{M_0}$ does not accept $\boldsymbol{M}$
$\Rightarrow$ $\boldsymbol{M}$ loops forever for input $\boldsymbol{M}$

$\Rightarrow$ $\boldsymbol{M}$ is in $\overline{\text{HALT}}$

Contradiction! So $\overline{\text{HALT}}$ is not r.e.

Exercises

Theorem

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

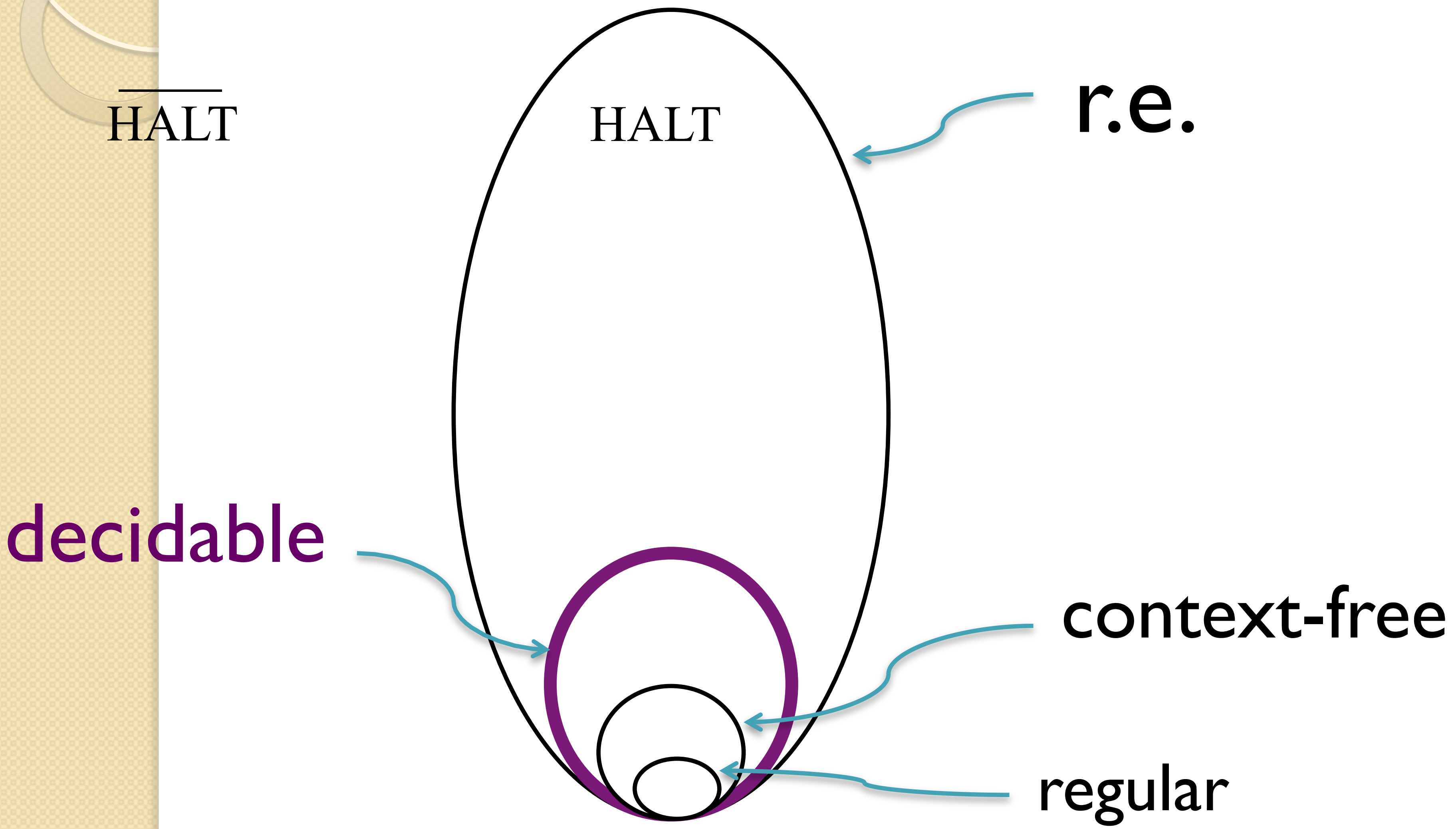
Theorem

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

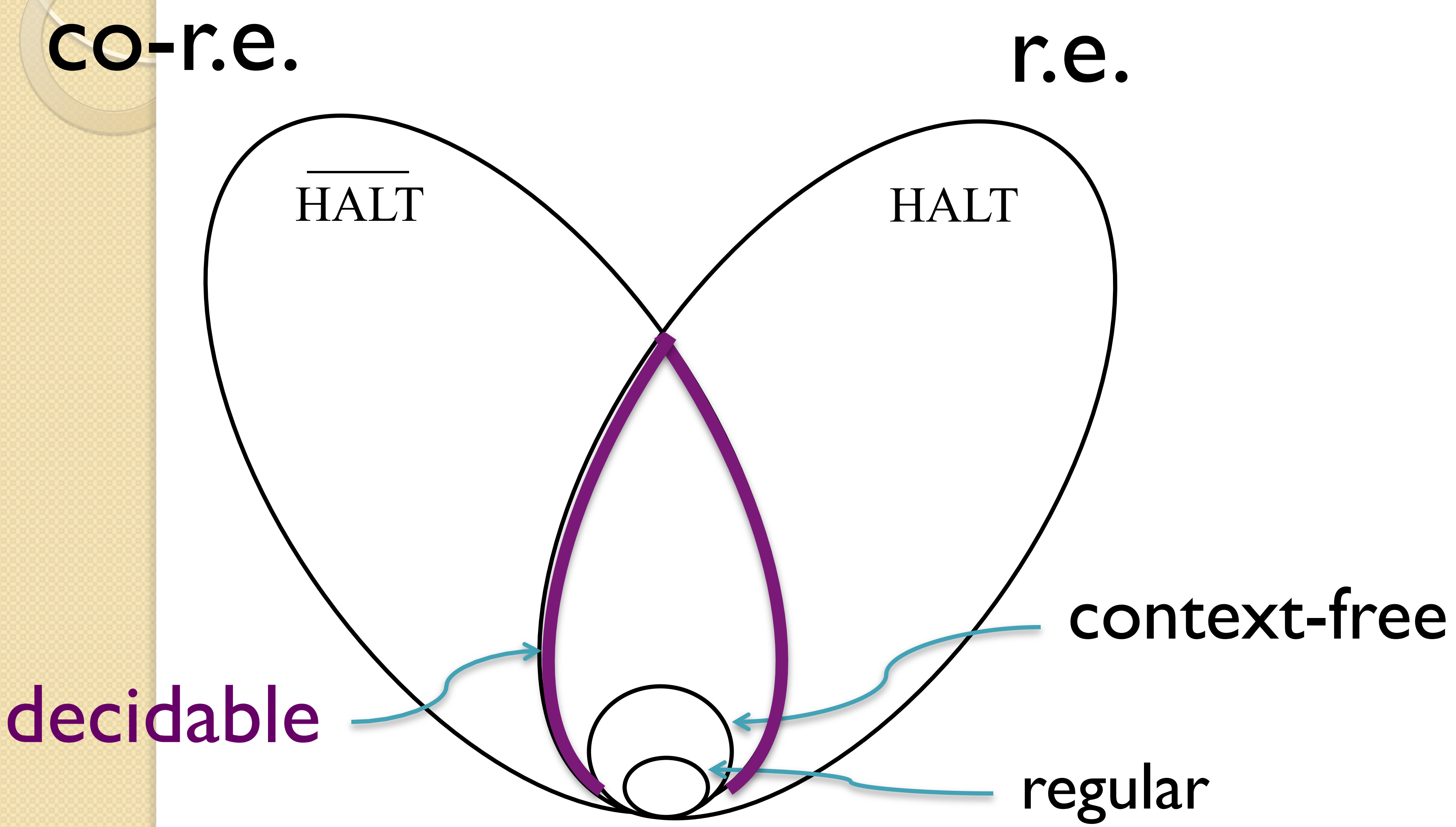
$\boldsymbol{x}$ is in $\boldsymbol{L}$ $\Leftrightarrow$ there exists $\boldsymbol{y}$ such that $\boldsymbol{P(x,y)}$.

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***.

Recursively enumerable languages



Recursively enumerable languages



Supplementary material

Formal Logical Systems

Slides by David Albrecht (2011)

Formal Logic System

A formal logic system is any mechanical procedure for producing formulas, called provable formulas.

- In other words in a formal logic system the set of provable formulae is computable.

Properties of Formal Logic Systems

- **Soundness**
  - Every formula that can be proved is true.
- **Completeness**
  - Every formula which is true is provable.
- **Decidability**
  - The set of provable formulae is decidable.

Gödel’s Incompleteness Theorem

For every formal logic system which contains arithmetic there exists a formula which is true but not provable.

Proof

- The number of computable subsets of the natural numbers is countable.
- Let $\mathbf{S_1}$, $\mathbf{S_2}$, … denote computable subsets of the natural numbers.
- Let

  D = {n: It is provable that "$n \notin S_n$"}
- Then, for some **k**, **D = $\mathbf{S_k}$**.
- It follows that "$k \notin S_k$" is true but not provable.

Turing Results

- If a formal logic system is sound and complete then it is decidable.
- Formal systems which contain arithmetic are not decidable.

  Proof:
  - Associate with each Turing Machine a formula, so that this formula is provable iff the corresponding Turing Machine halts.
  - Therefore the system is decidable iff the halting problem is decidable.

Revision

- Know the definition of recursively enumerable languages.
- Know about enumerators and their relationship with r.e. languages.
- Know a language that is r.e. but not decidable, and be able to prove this.

Reading:

- Sipser, pp 170, 209-211.

Preparation:

- Sipser, pp. 275-286.