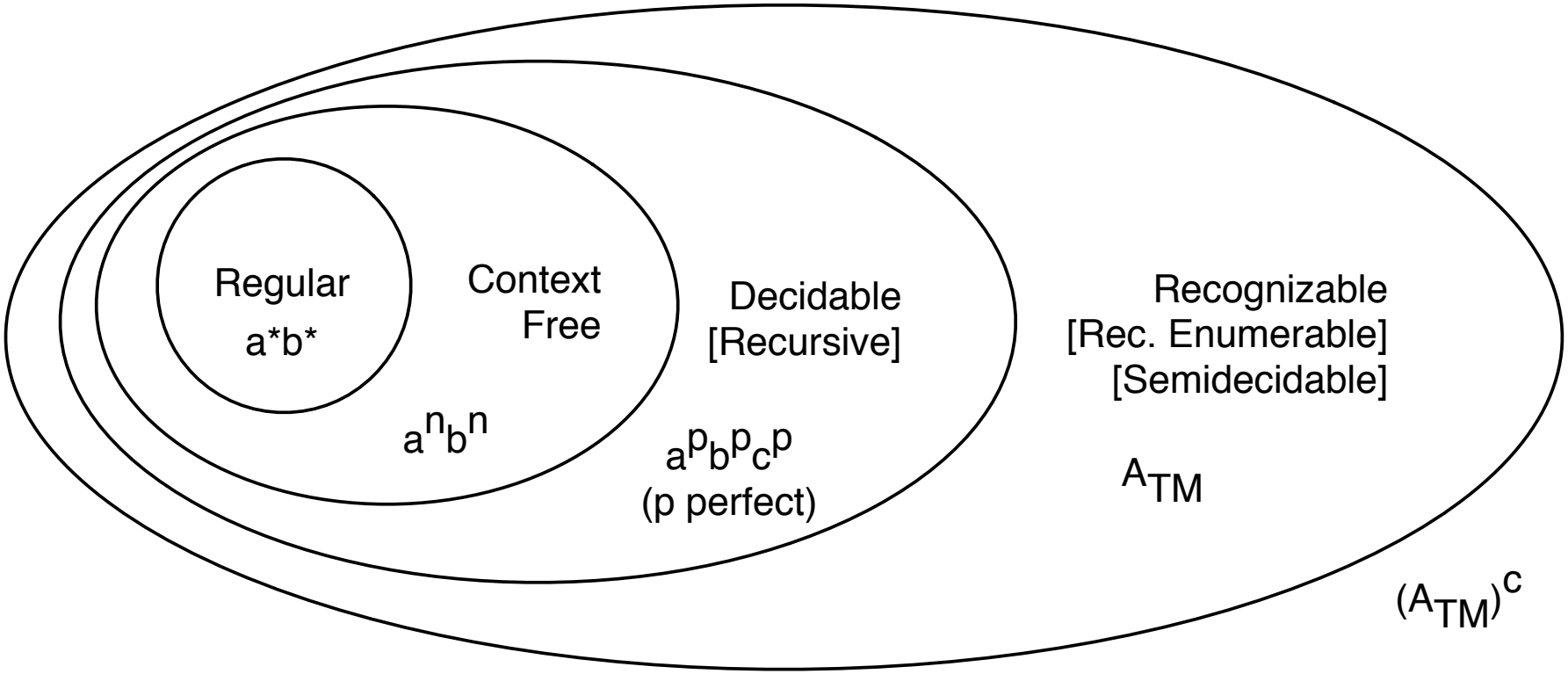
A mathematician and an engineer are on a desert island. They find two palm trees with one coconut each. The engineer climbs up one tree, gets the coconut, eats. The mathematician climbs up the other tree, gets the coconut, climbs the other tree and puts it there. "Now we've reduced it to a problem we know how to solve."

Undecidability & Reductions

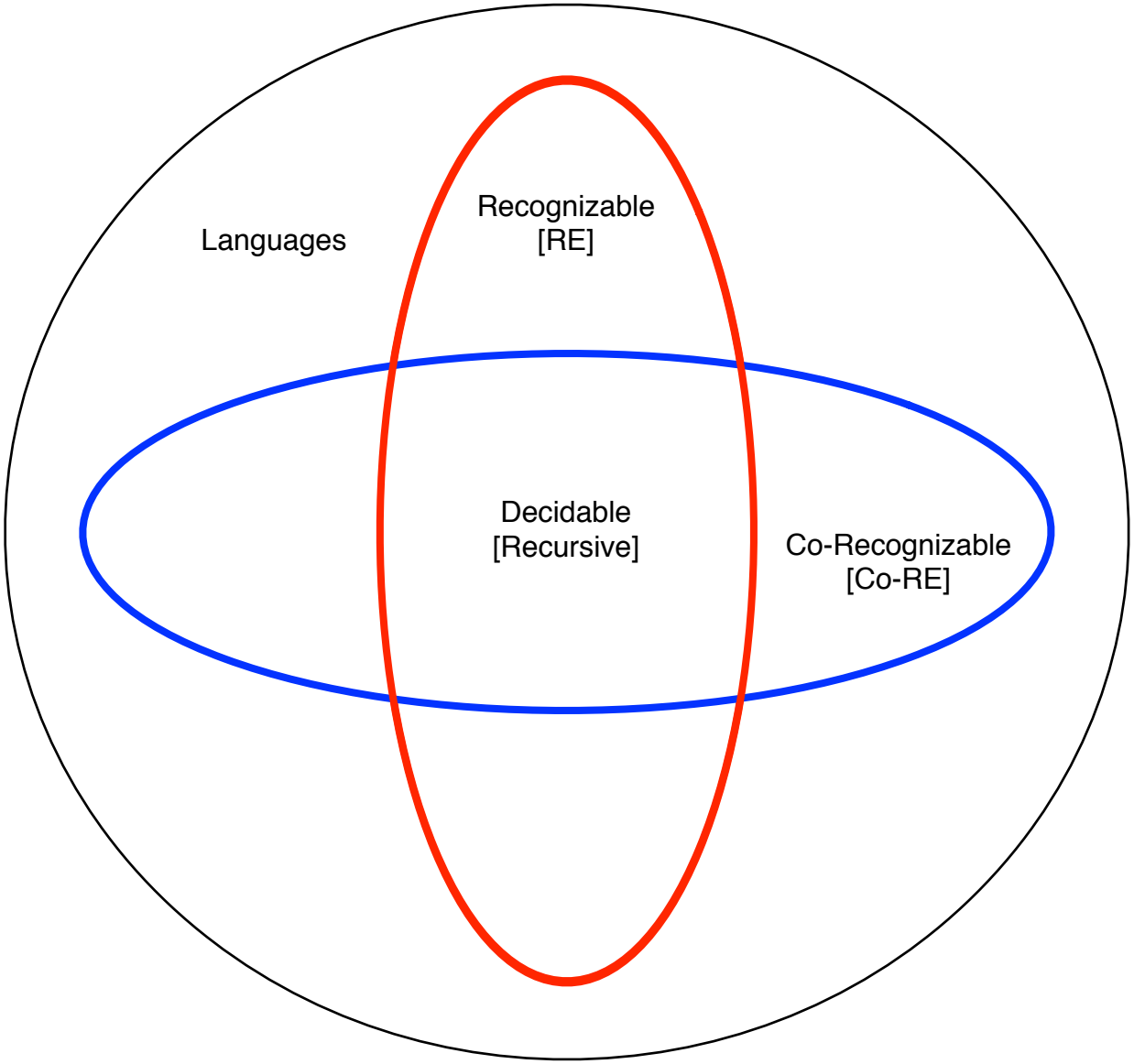
CS 81: Computability and Logic

November 18, 2010

Recall


Regular
a*b*
Context
Free
$a^n b^n$
Decidable
[Recursive]
$a^p b^p c^p$
(p perfect)
Recognizable
[Rec. Enumerable]
[Semidecidable]
A_{TM}
$(A_{TM})^c$

Recall


Languages
Recognizable
[RE]
Decidable
[Recursive]
Co-Recognizable
[Co-RE]

Warmup

Theorem. Every recognizable
[Recursively Enumerable] language
can be enumerated by a Turing Machine.

Undecidability

- Last time, we saw that not all languages are decidable, or even recognizable!

- Is the following language decidable? Recognizable?

L := { <M> | TM M does not accept <M> }

Reductions

- I want to phone Alice.
- Bob has Alice's phone number, if anyone does.
- Thus, I can reduce the problem of phoning Alice to the problem of getting the number from Bob.

- We say that problem P reduces to problem Q ($P \leq Q$) if solving P is "easier than" (no harder than) solving Q.

Phoning Alice ≤ Getting # from Bob.

Consequences of A Reduction

Phoning Alice $\leq$ Getting # from Bob

What consequences follow if:

- ✓ It's easy to talk to Bob?
- ✓ Bob is on Safari and unreachable?
- ✓ Alice has no phone?

In general, what consequences follow from $P \leq Q$?

General Reduction

If we want to show

$$E_{asy} \leq H_{ard}$$

That is, prove that E_{asy} is no harder than H_{ard}:

Do we need to show that:

- ✓ If I could solve E_{asy}, then I could solve H_{ard}?
- ✓ If I could solve H_{ard}, then I could solve E_{asy}?

Reduction for Mathematicians (and other people who like to solve problems)

- Typical approach: show that

$$U_{nsolved} \leq S_{olved}$$

- That is, prove that $U_{nsolved}$ is no harder than S_{olved}
- That is, if we know how to solve S_{olved}, we can then solve $U_{nsolved}$.

Reduction for CS81'ers (and other people who like to prove that problems are hard)

- Typical approach: show that

$$H_{ard} \leq U_{nknown}$$

- That is, prove that H_{ard} is easier than U_{nknown}
- That is, if we knew how to solve U_{nknown}, we could solve H_{ard}.
- In the case of Turing Machines,
  - give an algorithm for H_{ard} using a subroutine for U_{nknown}.
  - or, an algorithm turning any input for H_{ard} into an input to U_{nknown} that has the same yes/no answer.

Recall: Halting is Undecidable

- The language

  $A_{TM} := \{ \langle M,w \rangle \mid M \text{ accepts } w \}$

  was directly shown to not be decidable.

- If we want to argue that

  $H := \{ \langle M,w \rangle \mid M \text{ a TM that halts given } w \}$

  is not decidable, do we want to show

  - $H \leq A_{TM}$?
  - $A_{TM} \leq H$?

More Examples

- $NE_{TM} = \{ \langle M \rangle \mid M \text{ accepts at least one } w \in \Sigma^* \}$
  - Show that $A_{TM} \leq NE_{TM}$.

- $E_{TM} = \{ \langle M \rangle \mid L(M) = \emptyset \}$
- $All_{TM} = \{ \langle M \rangle \mid L(M) = \Sigma^* \}$
- Accepts-s = $\{ \langle M \rangle \mid M \text{ accepts s} \}$
- Regular = $\{ \langle M \rangle \mid L(M) \text{ is regular} \}$

Functional vs. Structural

- Structural questions (often, but not always, decidable):
  - Does this TM have **more than 500 control states**?
  - Does this TM ever use **more than 300 cells of tape on a given input**?
  - Does this TM ever write the symbol a ?
- Functional questions (rarely decidable):
  - Does this TM accept **the empty language**?
  - Does this TM accept **a finite language**?
  - Does this TM accept **a regular language**?
  - In general:

    Given a TM T, does L(T) have property P?

    Given a TM-recognizable L, does L have property P?

Rice's Theorem

No nontrivial, functional property of Turing Machines is decidable.

That is, no nontrivial property of recognizable languages is decidable.

Consequences?

"That is, no nontrivial property of recognizable languages is decidable."

- $NE_{TM} = \{ \langle M \rangle \mid M \text{ accepts at least one } w \in \Sigma^* \}$
- $E_{TM} = \{ \langle M \rangle \mid L(M) = \varnothing \}$
- $All_{TM} = \{ \langle M \rangle \mid L(M) = \Sigma^* \}$
- Accepts-s = { <M> | M accepts s }
- Regular = { <M> | L(M) is regular }

Proof (Setup)

- Given a nontrivial property P(L).
  - Assume $\neg P(\emptyset)$.
  - Pick a TM K_P such that $P(L(K_P))$ holds.
- Plan:
  - Show that $A_{TM} \leq P$.
  - I.e., from a P-decider (and K_P) build an acceptance-decider.

Reduction

Assume $\neg P(\varnothing)$.

Pick a TM K_P such that $P(L(K_P))$ holds.

- Given $\langle M,w \rangle$:
  1. Construct M', which does the following:
     a. Save its input x to an auxiliary tape
     b. Run M on w.
     c. If M accepts w, run K_P on x. Otherwise reject.
  2. Run the P-checker on M'.

The CS70 Grader Permanent Employment Theorem

The language

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \}$$

is undecidable.

[The language

$$EQ_{C++} = \{ \langle p1.cpp, p2.cpp \rangle \mid L(p1) = L(p2) \}$$

is also undecidable.]

The Full Employment Theorem for Compiler Writers

There is no perfect size-optimizing compiler.

Garbage Collection

- Java, Python, Haskell, Scheme, etc., all rely on garbage collection to deallocate unused memory.
  - At any point during execution, a piece of data is live it it will be used in the future, and otherwise dead or garbage.
  - A garbage collector detects and deallocates garbage.
  - Perfect garbage collection is undecidable.