# CS21
# Decidability and Tractability

Lecture 13

February 5, 2018

# Outline

- a non-RE and non-co-RE language
- the Recursion Theorem

- Gödel Incompleteness Theorem

# Dec. and undec. problems

- the boundary between decidability and undecidability is often quite delicate
  - seemingly related problems
  - one decidable
  - other undecidable

- We will see two examples of this phenomenon next.

# Dec. and undec. problems

- two problems regarding Context-Free Grammars:
  - does a CFG generate all strings:

    $ALL_{CFG}$ = {<G> : G is a CFG and L(G) = $\Sigma^*$}
  - CFG emptiness:

    $E_{CFG}$ = {<G> : G is a CFG and L(G) = $\emptyset$}

- Both decidable? both undecidable? one decidable?

# Dec. and undec. problems

**Theorem**: $E_{CFG}$ is decidable.

**Proof**:

– observation: for each nonterminal A, the set

$$S_A = \{w : A \Rightarrow^* w\}$$

is non-empty iff there is some rule:

$$A \rightarrow \mathbf{x}$$

and $\forall$ non-terminals B in string x, $S_B \neq \emptyset$

# Dec. and undec. problems

**Proof**:

- on input <G>
- mark all terminals in G
- repeat until no new non-terminals get marked:
  - if there is a production $A \rightarrow x_1 x_2 x_3 \ldots x_k$
  - and each symbol $x_1, x_2, \ldots, x_k$ has been marked
  - then mark A
- if S marked, reject ($G \notin E_{CFG}$), else accept ($G 2 E_{CFG}$).
- terminates? correct?

# Dec. and undec. problems

**Theorem**: $ALL_{CFG}$ is undecidable.

**Proof**:

- reduce from co-$A_{TM}$ (i.e. show co-$A_{TM} \leq_m ALL_{CFG}$)
- what should f(<M, w>) produce?
- Idea:
  - produce CFG G that generates all strings that are ***not*** accepting computation histories of M on w

# Dec. and undec. problems

**Proof**:

– build a NPDA, then convert to CFG

– want to accept strings not of this form,

$$\#C_1\#C_2\#C_3\#...\#C_k\#$$

plus strings of this form but where

- $C_1$ is not the start config. of M on input w, or
- $C_k$ is not an accept. config. of M on input w, or
- $C_i$ does not yield in one step $C_{i+1}$ for some i

# Dec. and undec. problems

**Proof**:

- our NPDA nondeterministically checks one of:
  - $C_1$ is not the start config. of M on input w, or
  - $C_k$ is not an accept. config. of M on input w, or
  - $C_i$ does not yield in one step $C_{i+1}$ for some i
  - input has fewer than two #'s
- details of first two?
- to check third condition:
  - nondeterministically guess $C_i$ starting position
  - how to check that $C_i$ doesn't yield in 1 step $C_{i+1}$ ?

# Dec. and undec. problems

**Proof**:

- checking:
  - $C_i$ does not yield in one step $C_{i+1}$ for some i
- push $C_i$ onto stack
- at #, start popping $C_i$ and compare to $C_{i+1}$
  - accept if mismatch away from head location, or
  - symbols around head changed in a way inconsistent with M's transition function.
- is everything described possible with NPDA?

# Dec. and undec. problems

**Proof**:

- Problem: cannot compare $C_i$ to $C_{i+1}$
- could prove in same way that proved

  $\{ww: w \in \Sigma^*\}$ not context-free
- recall that

  $\{ww^R: w \in \Sigma^*\}$ is context-free
- free to tweak construction of G in the reduction
- solution: write computation history:

  $\#C_1\#C_2^R\#C_3\#C_4^R...\#C_k\#$

# Dec. and undec. problems

**Proof**:

– f(<M, w>) = <G> equiv. to NPDA below:

on input x, accept if not of form:

$$\#C_1\#C_2^R\#C_3\#C_4^R...\#C_k\#$$

- accept if $C_1$ is the not the start configuration for M on input w
- accept if check that $C_i$ does not yield in one step $C_{i+1}$
- accept if $C_k$ is not an accepting configuration for M

- is f computable?
- YES maps to YES?

  $\langle M, w\rangle \in \text{co-}A_{TM} \Rightarrow$
  $f(M, w) \in ALL_{CFG}$
- NO maps to NO?

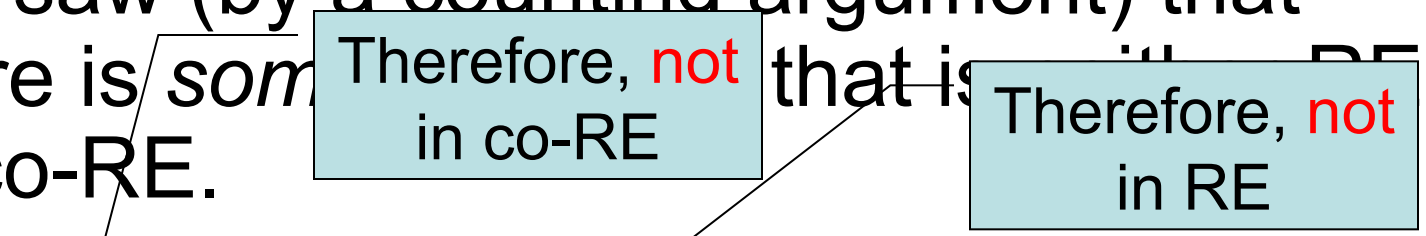
  $\langle M, w\rangle \notin \text{co-}A_{TM} \Rightarrow$
  $f(M, w) \notin ALL_{CFG}$

# Beyond RE and co-RE

- We saw (by a counting argument) that there is *some* language that is neither RE or co-RE.
- We will prove this for a natural language:

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

- Recall:
  - $A_{TM}$ is undecidable, but RE
  - co-$A_{TM}$ is undecidable, but coRE



# Beyond RE and co-RE

**Theorem**: $EQ_{TM}$ is neither RE nor coRE.

**Proof:**

- not RE:
  - reduce from co-$A_{TM}$ (i.e. show co-$A_{TM} \leq_m EQ_{TM}$)
  - what should f(<M, w>) produce?
- not co-RE:
  - reduce from $A_{TM}$ (i.e. show $A_{TM} \leq_m EQ_{TM}$)
  - what should f(<M, w>) produce?

# Beyond RE and co-RE

**Proof** ($A_{TM} \leq_m EQ_{TM}$)

– f(<M, w>) = <$M_1$, $M_2$> described below:

TM $M_1$: on input x,

• accept

TM $M_2$: on input x,

• simulate M on input w

• accept if M accepts w

•YES maps to YES?

$\langle M, w\rangle \in A_{TM} \Rightarrow$
$L(M_1) = \Sigma^*, L(M_2) = \Sigma^*, \Rightarrow$
$f(\langle M, w\rangle) \in EQ_{TM}$

• NO maps to NO?

$\langle M, w\rangle \notin A_{TM} \Rightarrow$
$L(M_1) = \Sigma^*, L(M_2) = \emptyset \Rightarrow$
$f(\langle M, w\rangle) \notin EQ_{TM}$

# Beyond RE and co-RE

**Proof** (co-$A_{TM} \leq_m EQ_{TM}$)

– f(<M, w>) = <$M_1$, $M_2$> described below:

TM $M_1$: on input x,

- reject

TM $M_2$: on input x,

- simulate M on input w
- accept if M accepts w

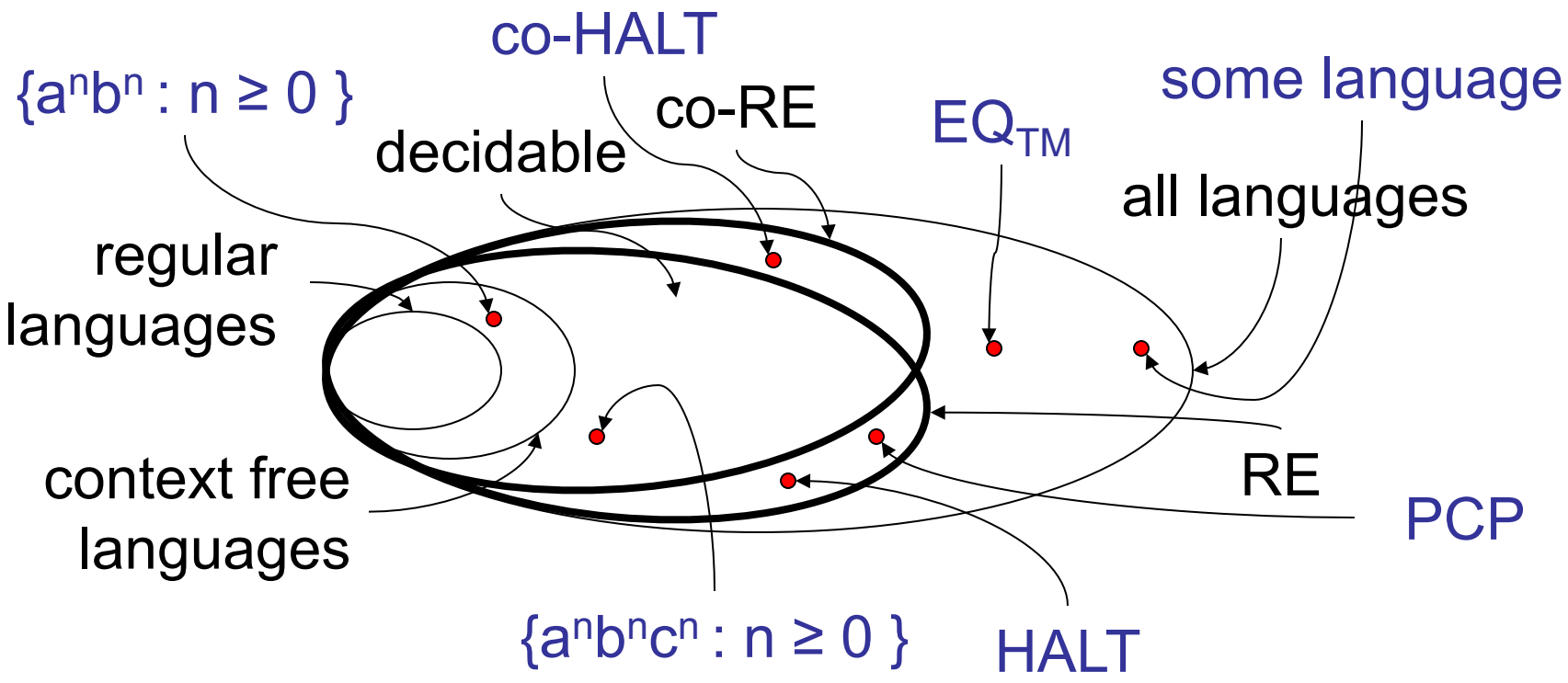
•YES maps to YES?

<M, w> $\in$ co-$A_{TM}$ $\Rightarrow$
$L(M_1) = \emptyset$, $L(M_2) = \emptyset$ $\Rightarrow$
f(<M, w>) $\in$ $EQ_{TM}$

• NO maps to NO?

<M, w> $\notin$ co-$A_{TM}$ $\Rightarrow$
$L(M_1) = \emptyset$, $L(M_2) = \Sigma^*$, $\Rightarrow$
f(<M, w>) $\notin$ $EQ_{TM}$

# Summary


co-HALT
{a^n b^n : n ≥ 0 }
co-RE
some language
EQ_TM
decidable
all languages
regular
languages
context free
languages
RE
PCP
{a^n b^n c^n : n ≥ 0 }
HALT

# The Recursion Theorem

- A very useful, and non-obvious, capability of Turing Machines:
  - in the course of computation, can print out a description of itself!
- how is this possible?
  - an example of a program that prints out self:

Print two copies of the following, the 2nd one in quotes:

"Print two copies of the following, the 2nd one in quotes:"

# The Recursion Theorem

- Why is this useful?
- Example: slick proof that $A_{TM}$ undecidable
  - assume TM M decides $A_{TM}$
  - construct machine M' as follows:

on input x,

- obtain own description <M'>
- run M on input <M', x>
- if M rejects, accept; if M accepts, reject.

if M' on input x:

- accepts, then M rejects <M', x>, but then M' does not accept!
- rejects, then M accepts <M', x>, but then M' accepts!

# The Recursion Theorem

- Lemma: there is a computable function

$$q:\Sigma^* \rightarrow \Sigma^*$$

  such that q(w) is a description of a TM $P_w$ that prints out w and then halts.

- Proof:
  - on input w, construct TM $P_w$ that has w hard-coded into it; output $<P_w>$

# The Recursion Theorem

- Warm-up: produce a TM SELF that prints out its own description.
- Two parts:
  - Part A:
    - output a description of B
    - pass control to B.
  - Part B:
    - prepend a description of A
    - done

# The Recursion Theorem

- Part A:
  - output a description of B
  - pass control to B.
- Part B:
  - prepend a description of A
  - done

Recall: q(w) is a description of a TM $P_w$ that prints out w and then halts.

Note: <A> = q(<B>)

| A |
| --- |
| • output <B> |

| B |
| --- |
| • read contents of tape |
| • apply q to it |
| • prepend* result to tape |

*combine with description on tape to produce a complete TM

# The Recursion Theorem

Note: <A> = q(<B>)

| A |
| --- |
| • output <B> |

| B |
| --- |
| • read contents of tape |
| • apply q to it |
| • prepend result to tape |

Recall: q(w) is a description of a TM $P_w$ that prints out w and then halts.

- watch closely as TM AB runs:
- A runs. Tape contents: <B>
- B runs. Tape contents: q(<B>)<B> ) <AB>
- AB is our desired machine SELF.

# The Recursion Theorem

**Theorem**: Let T be a TM that computes fn:

$$t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

There is a TM R that computes the fn:

$$r: \Sigma^* \rightarrow \Sigma^*$$

defined as $r(w) = t(w, \langle R \rangle)$.

- This allows "obtain own description" as valid step in TM program
  - first modify TM so that it takes an additional input (that is own description); use at will

# The Recursion Theorem

**Theorem**: Let T be a TM that computes fn:

$$t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

There is a TM R that computes the fn:

$$r: \Sigma^* \rightarrow \Sigma^*$$

defined as $r(w) = t(w, \langle R \rangle)$.

Proof outline: TM R has 3 parts

- Part A: output description of BT
- Part B: prepend description of A
- Part "T": run TM T

# The Recursion Theorem

Proof details: TM R has 3 parts

Part A: output description of BT

- <A> = q(<BT>)

Part B: prepend description of A

- read contents of tape <BT>
- apply q to it q(<BT>) = <A>
- prepend to tape <ABT>

Part "T": run TM T

- 2nd argument on tape is description of R

# Gödel Incompleteness Theorem

# Background

- Hilbert's program (1920's):
  - formalize mathematics in axiomatic form
  - derive all true statements "mechanically" from initial axioms
  - would put mathematicians out of business!
  - very influential proposal

- to start: try for all true statements about the natural numbers ("number theory")

# Background:

- Kurt Gödel (1931): it is not possible!
- no formalization of number theory can prove all true statements
- stunning result
- considered one of greatest 20$^{th}$ century achievements in mathematics

# Background

- We will prove using:
  - RE languages and non-RE languages
  - reductions
- Idea:
  - set of all theorems is RE
  - set of all true statements is not RE
- This kind of proof of Gödel's result attributed to Turing (1937).