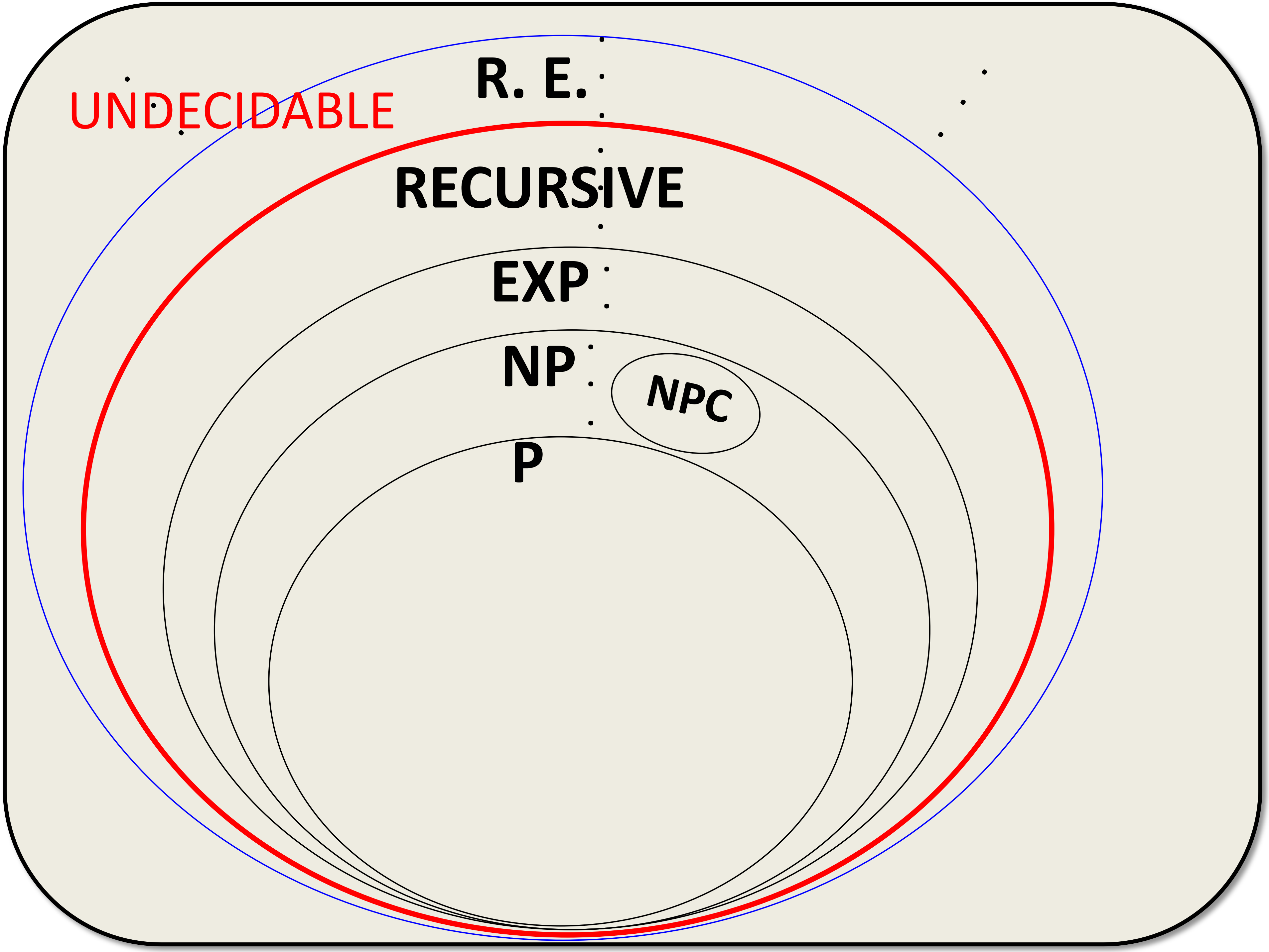
Undecidability and Rice's Theorem

Lecture 25, Dec 6

CS 374, Fall 2018


UNDECIDABLE
R. E.
RECURSIVE
EXP
NP
NPC
P

Recap: Universal TM U

We saw a TM *U* such that

$L_u = L(U) = \{ \langle M \rangle \# w \mid M \text{ accepts } w \}$

Thus, *U* is a stored-program computer.

It reads a program *<M>* and executes it on data *w*

L_u is r.e.

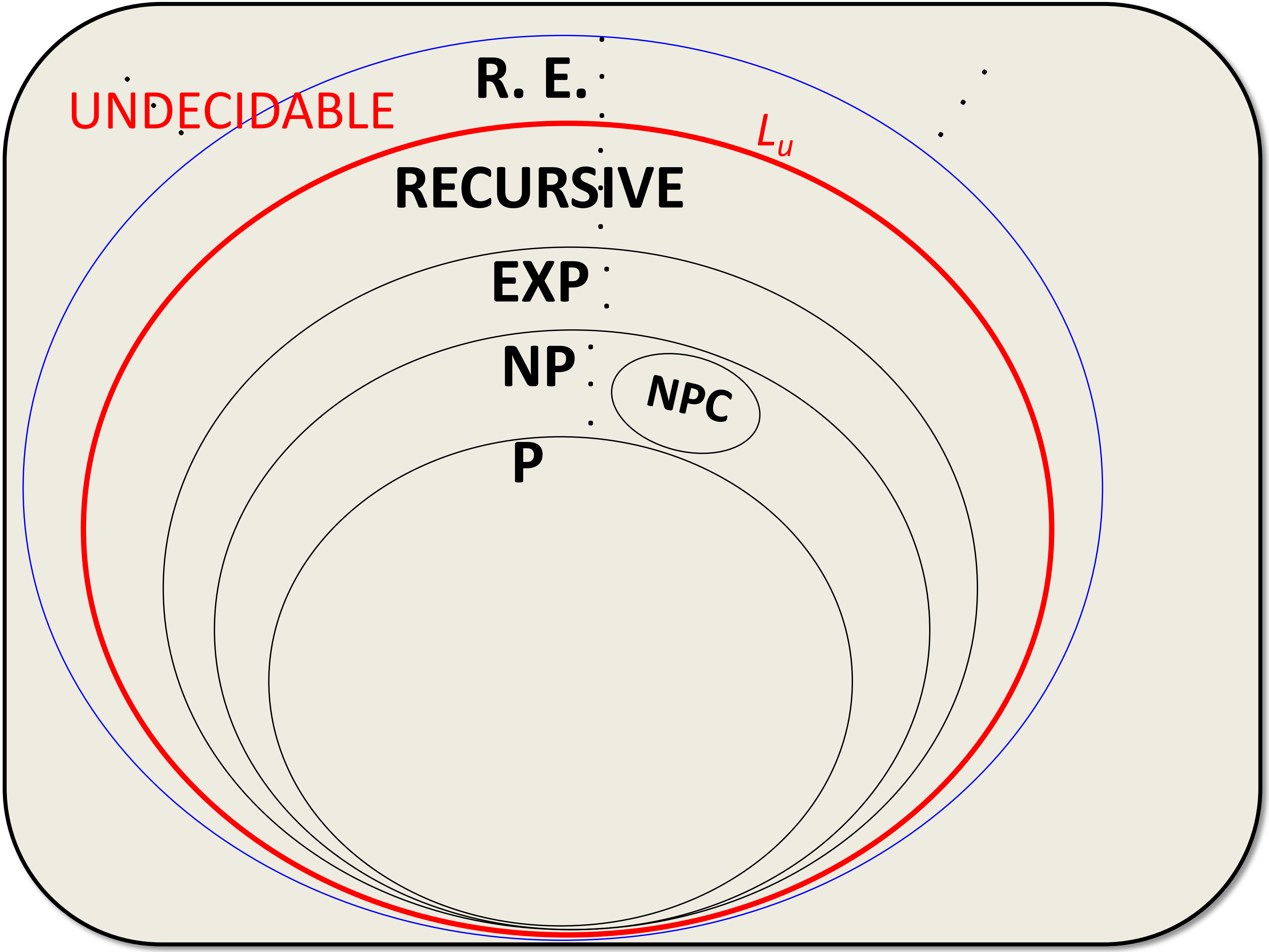
Recap: Universal TM U

L_u = { <*M*> # *w* | *M* accepts *w*} is r.e.

We proved the following:

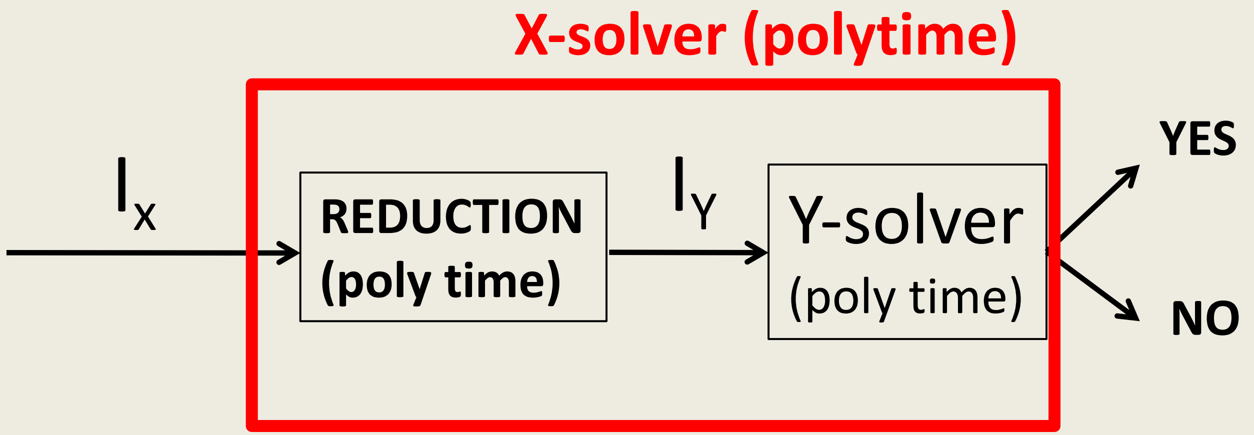
Theorem: L_u is undecidable (i.e, not recursive)

No "algorithm" for L_u


UNDECIDABLE
R. E.
L_u
RECURSIVE
EXP
NP
NPC
P

Polytime Reductions

$X \le_p Y$ "X reduces to Y in polytime"

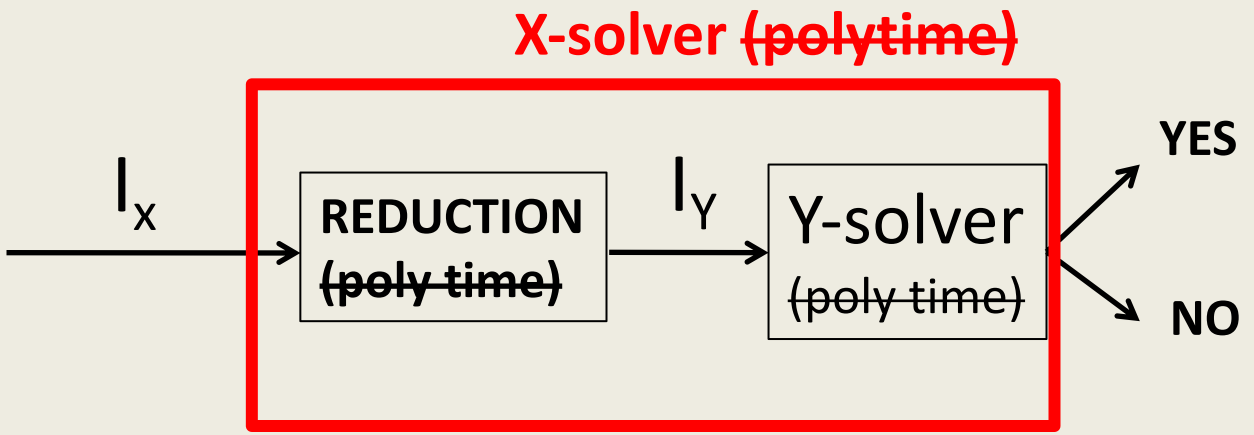


If Y can be decided in poly time, then X can be decided in poly time

If X can't be decided in poly time, then Y can't be decided in poly time

~~Polytime~~ Reductions

X ≤ Y “X reduces to Y ~~in polytime~~”

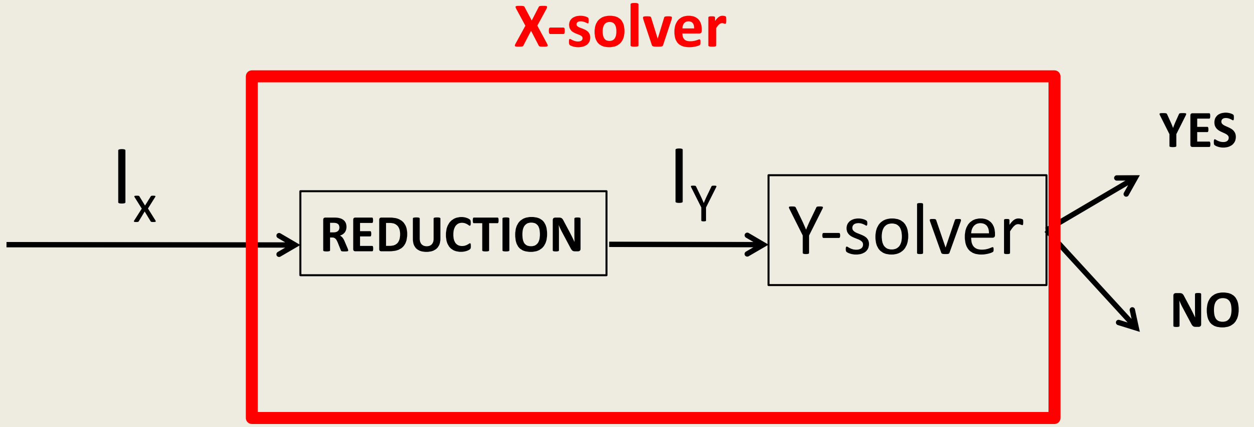


If Y can be decided ~~in poly time~~, then X can be decided ~~in poly time~~

If X can’t be decided ~~in poly time,~~ then Y can’t be decided ~~in poly time~~

Reduction

$X \leq Y$ "X reduces to Y"



If Y can be decided, then X can be decided.

If X can't be decided, then Y can't be decided

Reduction

$X \leq Y$ "X reduces to Y"

X-Solver(I_X) {
- Run reduction algorithm to create instance I_Y from I_X
- Return output of Y-Solver(I_Y)

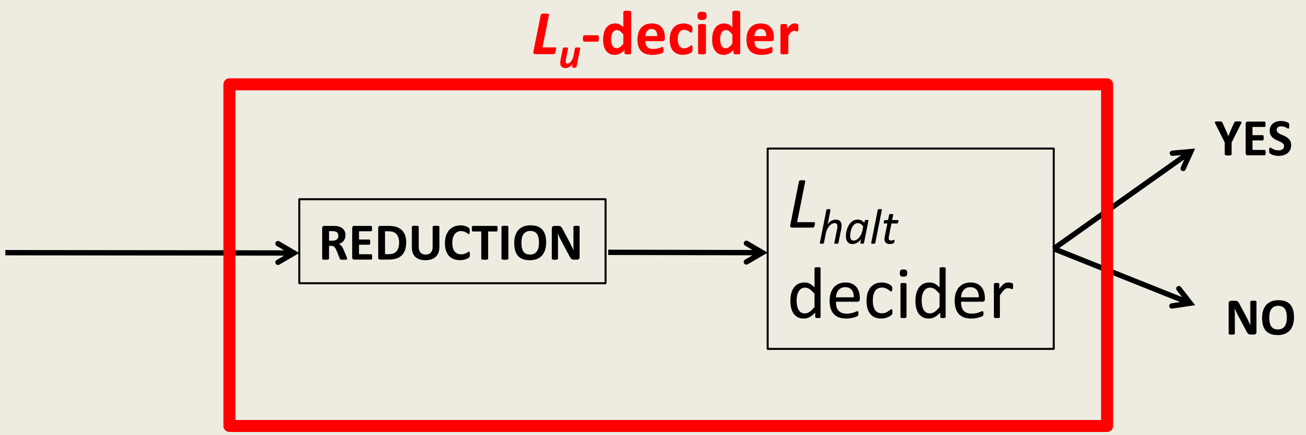
}

Using Reductions

- Once we have some seed problems such as L_d and L_u we can use reductions to prove that more problems are undecidable

Halting Problem

- Does given *M* halt when run on *blank input*?
- L_{halt} = {<*M*> | *M* halts when run on blank input}
- Show L_{halt} is undecidable by showing $L_u \leq L_{halt}$



What are input and output of the reduction?

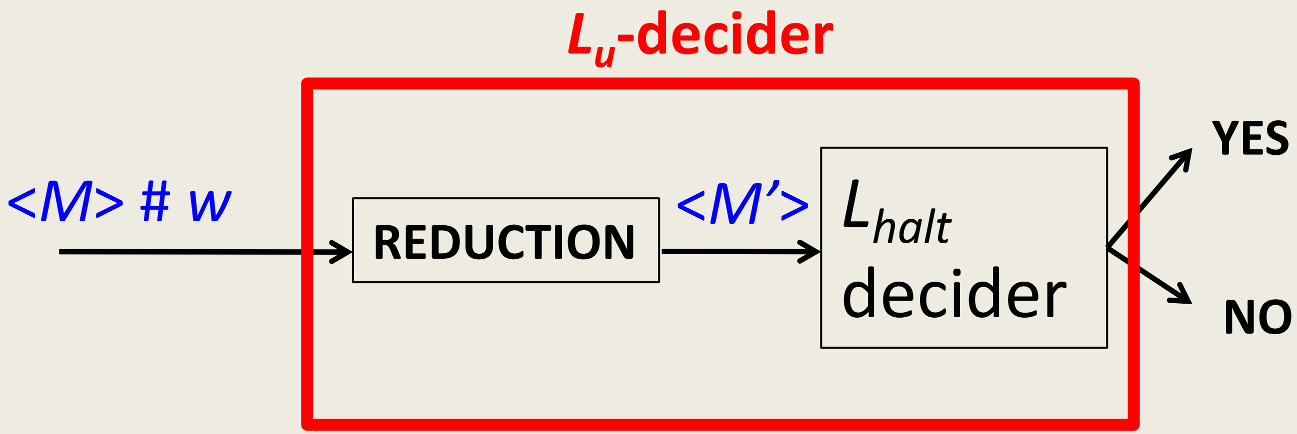
A different version of HALT

L_{halt} = { <M>#w | M halts on w }

Easier to show that this version of L_{halt} is undecidable by showing $L_u \leq L_{halt}$

Why?

$L_u \leq L_{halt}$



Need: *M'* halts on blank input iff *M*(*w*) accepts

<u>TM *M'*</u>

const *M*

const *w*

run *M*(*w*) and halt if it accepts

The REDUCTION **doesn't run *M* on *w***. It produces code for *M'* !

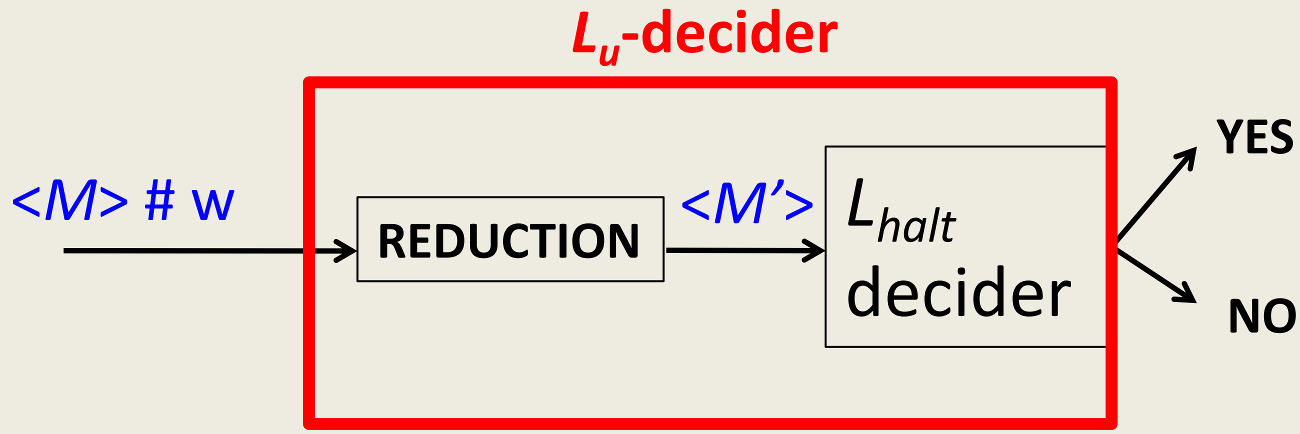
Example

- Suppose we have the code for a program isprime() and we want to check if it accepts the number 13
- The reduction creates new program to give to decider for L_{halt}: note that the reduction only creates the code, does not run any program itself.

```
main() {
        If (isprime(13)) then
            QUIT
        else
            LOOP FOREVER
      }

 boolean isprime(int i) {
    …
   }
```

$L_u \leq L_{halt}$



Need: *M'* halts on blank input iff *M*(*w*) accepts

<u>TM *M'*</u>

const *M*

const *w*

run *M*(*w*) and halt if it accepts

Correctness: L_u-decider say "yes" iff *M'* halts on blank input

iff *M*(*w*) accepts

iff <*M*>#*w* is in L_u

More reductions about languages

- We'll show other languages involving program behavior are undecidable:
- $L_{374} = \{\langle M \rangle \mid L(M) = \{0^{374}\}\}$
- $L_{\neq\emptyset} = \{\langle M \rangle \mid L(M) \text{ is nonempty}\}$
- $L_{pal} = \{\langle M \rangle \mid L(M) = \text{palindromes}\}$
- *many many* others

$L_{374} = \{\langle M \rangle \mid L(M) = \{0^{374}\}\}$ is undecidable

- Given a TM *M*, telling whether it accepts only the string 0^{374} is not possible
- Proved by showing $L_u \leq L_{374}$

M': constants: *M*, *w*

On input x,
  0. if $x \neq 0^{374}$, reject
  1. if $x = 0^{374}$, then run *M*(*w*)
     accept x iff *M*(*w*) ever accepts *w*

What is *L*(*M'*) ?
- If *M*(*w*) accepts, *L*(*M'*) = $\{0^{374}\}$
- If *M*(*w*) doesn't *L*(*M'*) = $\emptyset$

Q: How does the reduction know whether or not *M*(*w*) accepts ?
A: It doesn't have to. It just *builds* (code for) *M'*.

$L_{374} = \{<M> \mid L(M) = \{0^{374}\}\}$ is undecidable

- Given a TM M, telling whether it accepts only the string 0^{374} is not possible
- Prove by showing $L_{halt} \leq L_{374}$
- Reduction: an algorithm, that given a program $<M>$ creates a new program $<M'>$ such that $L(M') = \{0^{374}\}$ iff M halts on blank input
- Why does this suffice?

$L_{374} = \{<M> \mid L(M) = \{0^{374}\}\}$ is undecidable

- Reduction: an algorithm, that given a program <M> creates a new program <M'> such that $L(M') = \{0^{374}\}$ iff M halts on blank input

```
M'(input x) {
    Run M();
    If (x == 0^374) accept
    else reject
}
```

Note that reduction only creates code for M' from code for M

L_{374} = {<M> | $L(M) = \{0^{374}\}$ } is undecidable

- What about $L_{accepts\text{-}374}$ = {<M> | M accepts 0^{374}}
- Is this easier?
  - in fact, yes, since L_{374} isn't even r.e., but $L_{accepts\text{-}374}$ is
  - but no, $L_{accepts\text{-}374}$ is not decidable either
- The same reduction works:
  - If $M(w)$ accepts, $L(M') = \{0^{374}\}$, so M' accepts 0^{374}
  - If $M(w)$ doesn't, $L(M') = \emptyset$, so M' doesn't accept 0^{374}
- More generally, telling whether or not a machine accepts any fixed string is undecidable

$L_{\neq\emptyset}$ = {<M> | $L(M)$ is nonempty} is undecidable

- Given a TM M, telling whether it accepts *any* string is undecidable
- Proved by showing $L_{halt} \leq L_{\neq\emptyset}$

$L_{\neq\emptyset}$ = {<*M*> | *L*(*M*) is nonempty} is undecidable

- Reduction: an algorithm, that given a program <M> creates a new program <M'> such that L(M') = $\{0^{374}\}$ iff M halts on blank input

```
M'(input x) {
    Run M();
    accept x;
}
```

L_{pal} = {<M> | $L(M)$ = palindromes} is undecidable

- Given a TM M, telling whether it accepts the set of palindromes is undecidable
- Proved by showing $L_{HALT} \leq L_{pal}$

L_{pal} = {<*M*> | *L*(*M*) = palindromes} is undecidable

- Reduction: an algorithm, that given a program <M> creates a new program <M'> such that L(M') = $\{0^{374}\}$ iff M halts on blank input

```
M'(input x) {
    Run M();
    If (x is a palindrome)
        accept;
    else
        reject;
}
```

Lots of undecidable problems about languages accepted by programs

- Given M, is $L(M)$ = {palindromes}?
- Given M, is $L(M) \neq \emptyset$?
- Given M, is $L(M) = \{0^{374}\}$ [redacted]
- Given M, does $L($[redacted]
- Given M, is [redacted]e}?
- Given [redacted]n any prime?
- [redacted] contain *any word?*
- [redacted]s $L(M)$ meet these formal specs?
- G[redacted], does $L(M) = \Sigma^*$?

UNDECIDABLE

Rice's Theorem

- *Q:* What can we decide about the languages accepted by programs?

A: NOTHING !

except "trivial" things

Properties of r.e. languages

- A *Property of r.e. languages* is a predicate P of r.e. languages.

  i.e., P: $\{L \mid L \text{ is r.e.}\} \rightarrow \{\text{true, false}\}$

Important: we are only interested in r.e languages

- Examples:
  - $P(L)$ = "L contains 0^{374}"
  - $P(L)$ = "L contains at least 5 strings"
  - $P(L)$ = "L is empty"
  - $P(L)$ = "$L = \{0^n 1^n \mid n \geq 0\}$"

Properties of r.e. languages

- A *Property of r.e. languages* is a predicate *P* of r.e. languages.

  i.e., P: {L | L is r.e.} → {true, false}

L = L(M) for some TM iff L is r.e by definition.

- We will thus think of a *Property of r.e. languages* as a set { <M> | *L*(*M*) satisfies predicate *P*}
- Note that each property P is thus a set of strings L(P) = { <M> | *L*(*M*) satisfies predicate *P*}
- **Question:** For which P is L(P) decidable?

Trivial Properties

- A property is *trivial* if either ***all*** r.e. languages satisfy it, or ***no*** r.e. languages satisfy it.
- { <M> | *L*(*M*) is r.e}.... why is this “trivial” ?
  - EVERY language accepted by an *M* is r.e. by def’n
- { <M> | *L*(*M*) is not r.e}.... why is this “trivial” ?
- { <M>| *L*(*M*) = Ø or *L*(*M*) ≠Ø}.... why “trivial”?
- Clearly, trivial properties are decidable
- Because if P is trivial then L(P) = Ø or L(P) = Σ*

Rice's Theorem

Every nontrivial property of r.e. languages is undecidable

So, there is virtually nothing we can decide about behavior (language accepted) by programs

Example: auto-graders don't exist (if submissions are allowed to run an arbitrary (but finite) amount of time).

Proof

- Let P be a non-trivial property
- Let $L(P)$ = { <M> | $L(M)$ satisfies predicate P}
- Show $L(P)$ is undecidable
- Assume Ø does not satisfy P
- Assume $L(M_1)$ satisfies P for some TM M_1

There must be at least one such *TM* (why?)

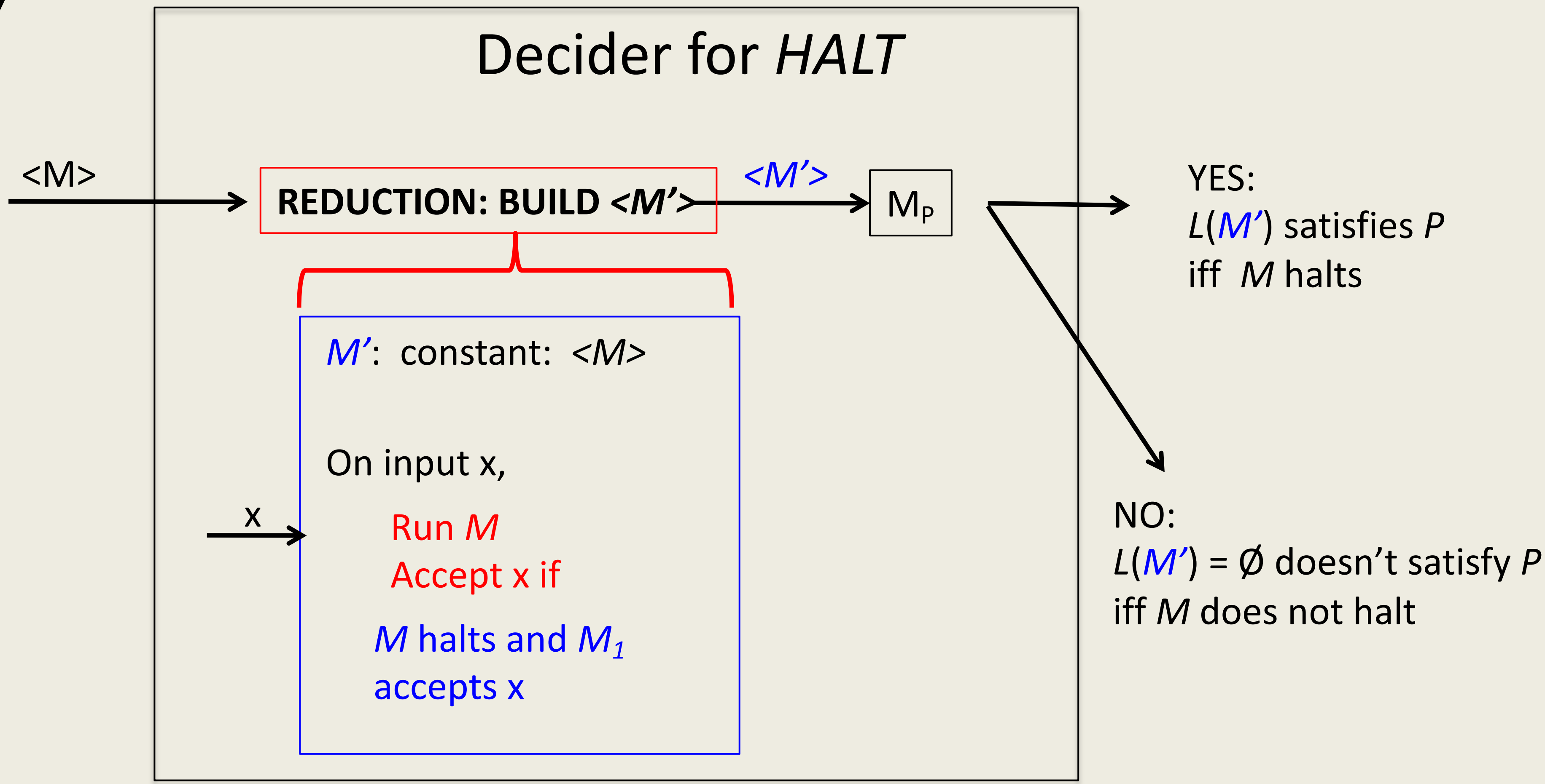
Proof

- Reduction: an algorithm, that given a program <M> creates a new program <M'> such that $L(M') = L(M_1)$ iff M halts on blank input

```
M'(input x) {
    Run M();
    Run M1(x);
}
```

If M halts on blank input $L(M') = L(M_1)$
else $L(M') = \emptyset$

If there is a decider M_P to tell if a TM accepts a language satisfying *P*...



If *M* doesn't halt then *L*(*M'*) = Ø

If *M* does halt then *L*(*M'*) = $L(M_1)$

Since *HALT* is not decidable, M_P *doesn't* exist, and *L(P)* is undecidable

What about assumption

- We assumed Ø does not satisfy *P*
- What if Ø does satisfy *P?*
- Then consider

  L(P') = { <*M*> | *L*(*M*) doesn't satisfy predicate *P*}
- Then Ø isn't in *L(P')*
- Show *L(P')* is undecidable
- So *L(P)* isn't either (by closure under complement)

*Properties of r.e Languages are **Not** properties of **programs/TMs***

- *P* is defined on languages, not the machines which might accept them.
- {<*M*> | *M* at some point moves its head left}

  is a property of the ***machine behavior***, not the **language accepted.**
- {<*A.py*> | program *A* has 374 lines of code}
- {<*A.py*> | *A* accepts "Hello World"}

  this really is a predicate on *L*(*A*)

Properties about TMs

- sometimes decidable:
  - { <M>| *M* has 374 states}
  - { <M>| *M* uses ≤ 374 tape cells on blank input}
    - $374 \times |\Gamma|^{32} \times |Q_M|$
  - { <M>| *M* never moves head to left}
- sometimes undecidable
  - { <M>| *M* halts on blank input}
  - { <M>| *M* on input “0110”, eventually writes “2”}

Today

- Quick recap – halting & undecidability
- Undecidability via reductions
- Rice’s theorem
- ICES

Final Thoughts

Theory of Computation and Algorithms are fundamental to Computer Science

Of immense pragmatic importance

Of great interest to mathematics

Of great interest to natural sciences (physics, biology, chemistry)

Of great interest to social sciences too!

Other TheoryAlgorithms Courses

- 473 (Theory 2) – every semester but not in Spring 19
- CS 574 Randomized algorithms (Fall'19?)
- CS 583 Approximation algorithms
- CS 579 Computational Complexity (Spring'18)
- CS498 Algorithms for BIG Data (Spring'19)
- **Special topics:** Algorithmic Game Theory, Data structures (Fall'19?), Computational Geometry, Algorithms for Big Data, Geometric Data Structures, Pseudorandomness, Combinatorial Optimization, …

Other "Theory ish" Courses

- Machine learning, statistical learning, reinforcement learning, graphical models, …
- Logic and formal methods
- Graph theory, combinatorics, …
- Coding theory, information theory, signal processing
- Computational biology

Final Thoughts

Grades are important but only in short term

Don’t be discouraged if you didn’t do well

Remember what you enjoyed learning and why

Learning is a life long process – more important to **learn how to learn**

Use your algorithmic/theory/analytical skills to differentiate yourself from other IT professionals

On Learning

Without seeking, truth cannot be known at all. It can neither be declared from pulpits , nor set down in articles nor in any wise be prepared and sold in packages ready for use. Truth must be ground for every man by himself out of its husk, with such help as he can get, but not without stern labour of his own.

--John Ruskin

Thanks!