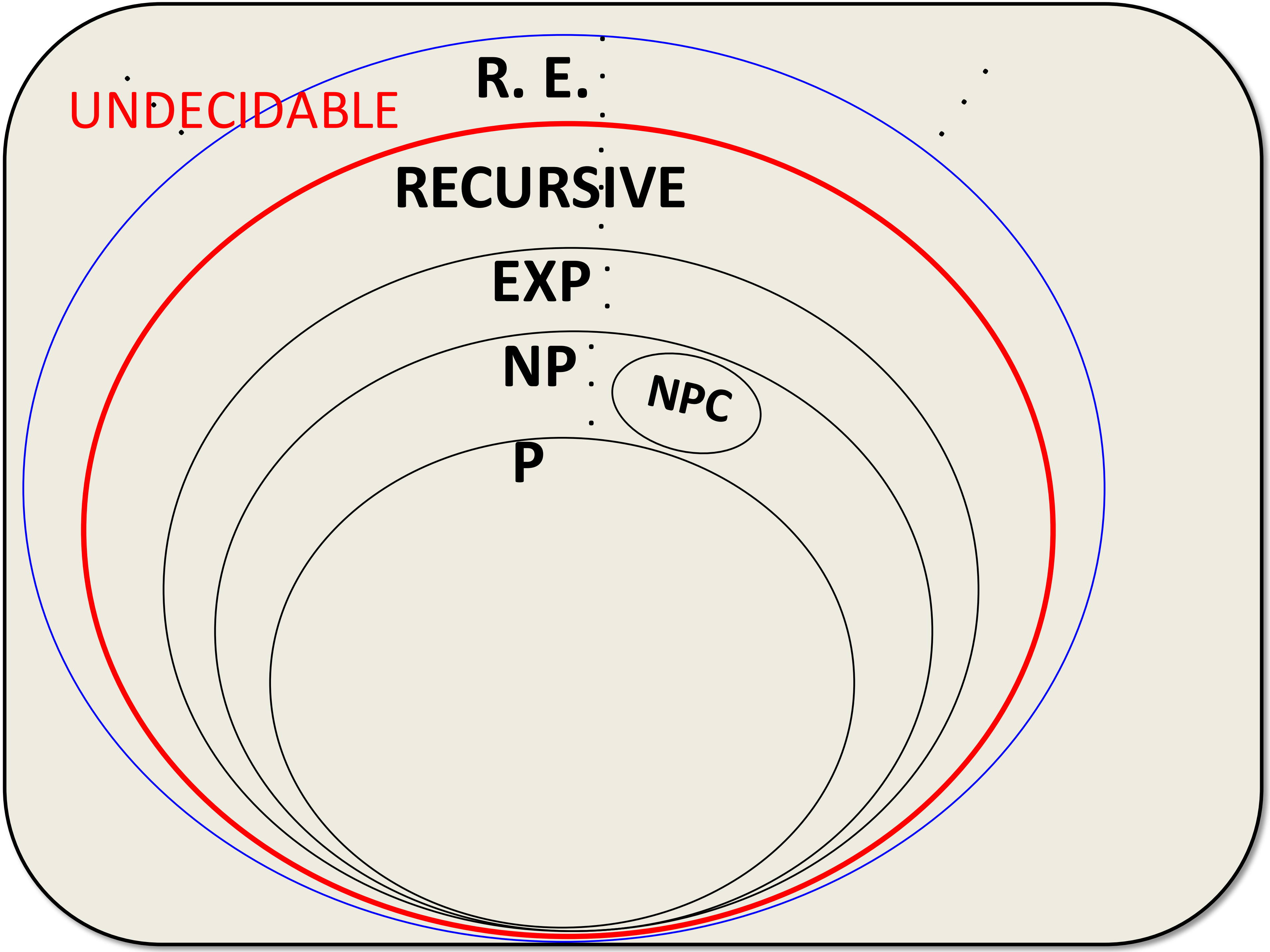
Undecidability and Rice's Theorem

Lecture 26, December 3

CS 374, Fall 2015


UNDECIDABLE
R. E.
RECURSIVE
EXP
NP
NPC
P

Recap: Universal TM U

We saw a TM U such that

$L(U) = \{ (z,w) \mid M_z \text{ accepts } w\}$

Thus, U is a stored-program computer.

It reads a program z and executes it on data w

$L(U) = \{ (z,w) \mid M_z \text{ accepts } w\}$ is r.e.

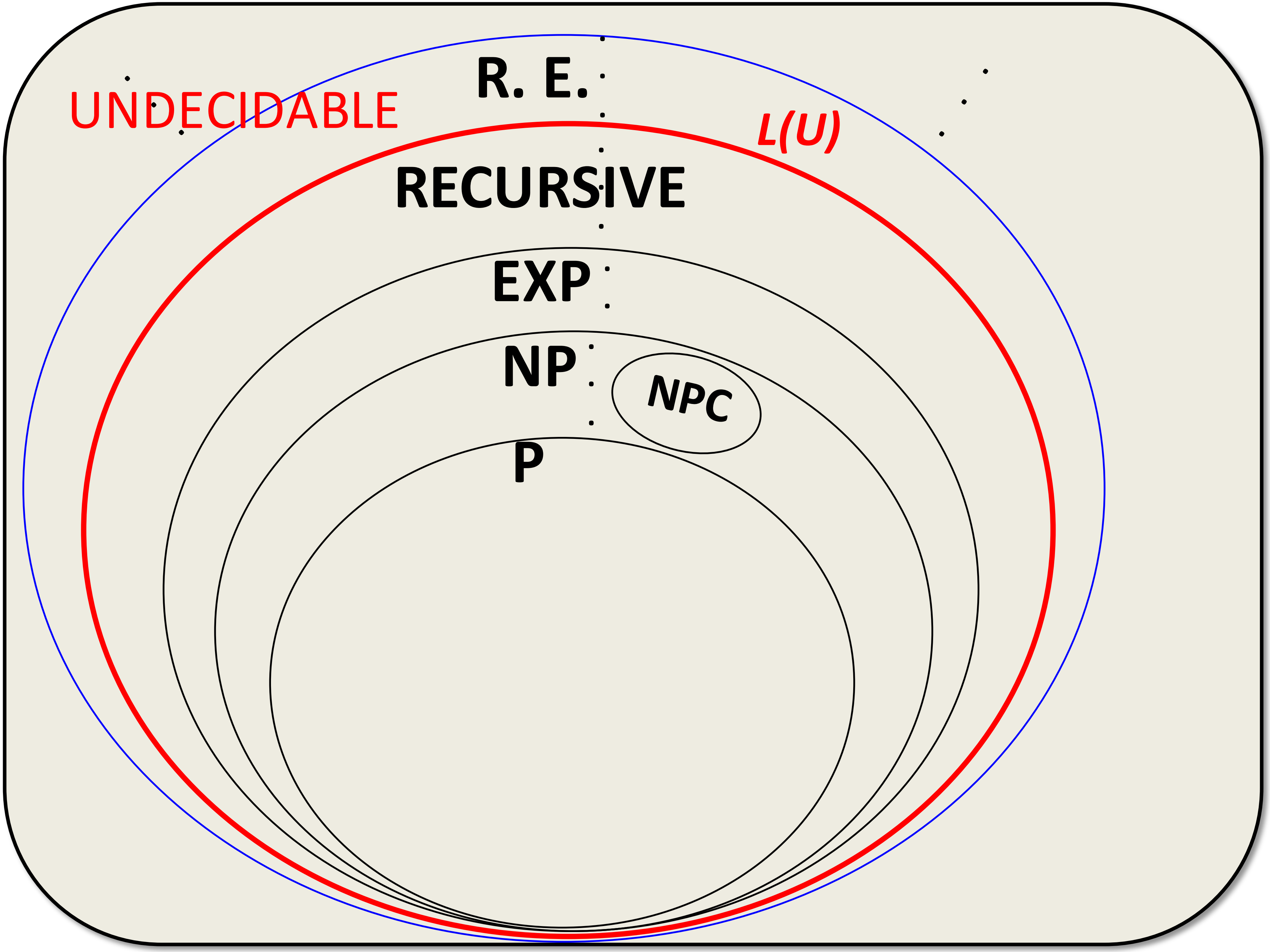
Recap: Universal TM U

$L(U) = \{ (z,w) \mid M_z$ accepts $w\}$ is r.e.

We proved the following:

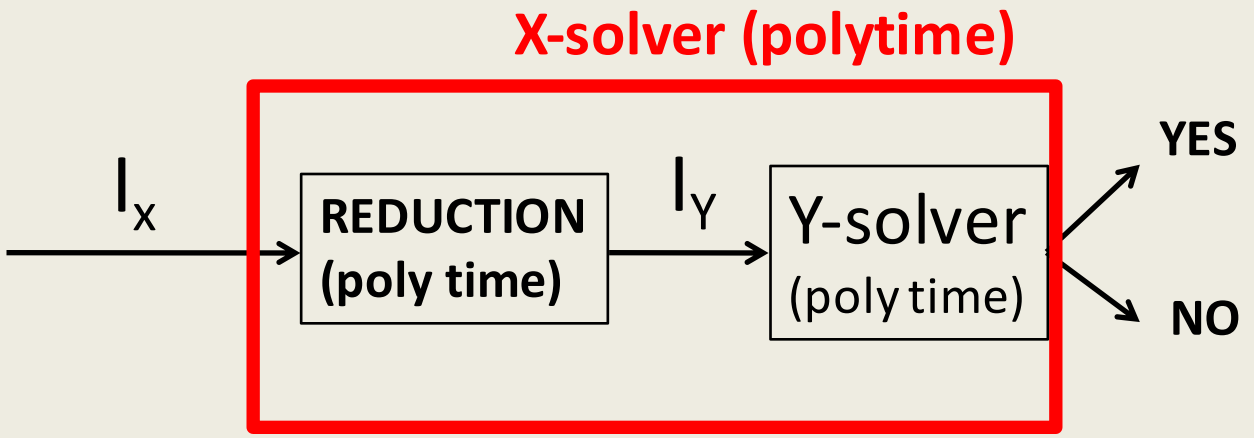
Theorem: *L(U)* is undecidable (i.e, not recursive)

No "algorithm" for L(U)


UNDECIDABLE
R. E.
L(U)
RECURSIVE
EXP
NP
NPC
P

Polytime Reductions

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

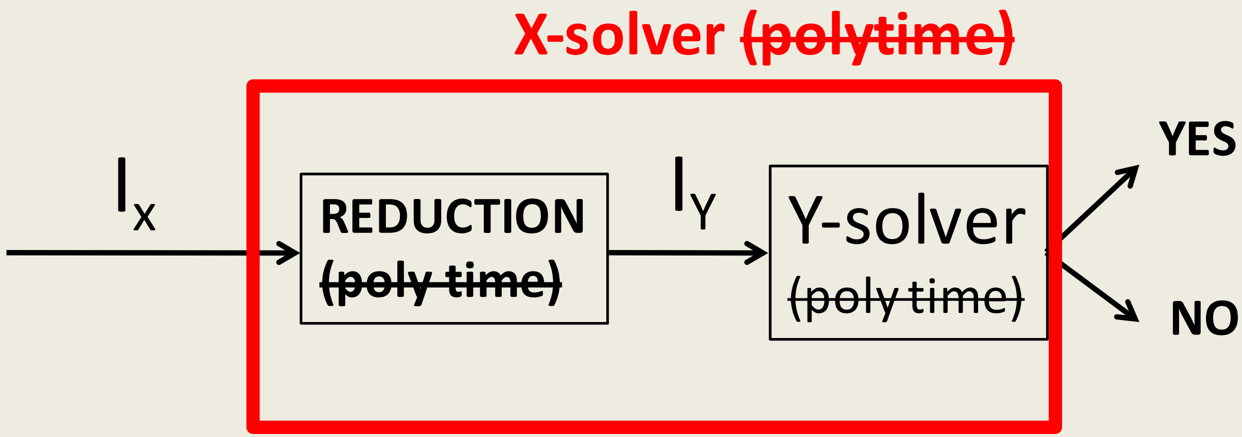


If Y can be decided in poly time, then X can be decided in poly time

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

~~Polytime~~ Reductions

$X \leq Y$ "X reduces to Y ~~in polytime~~"

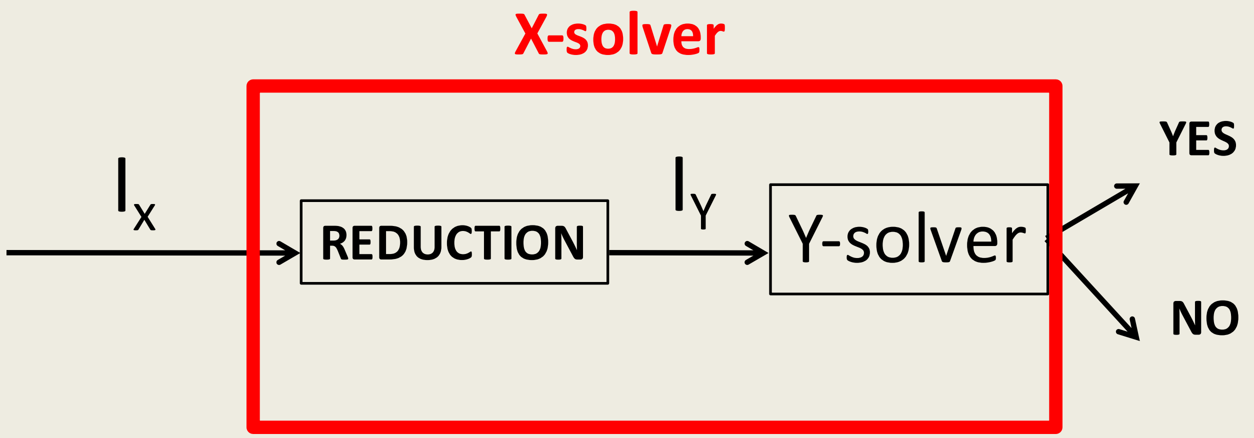


If Y can be decided ~~in poly time~~, then X can be decided ~~in poly time~~

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

Reduction

X ≤ Y   "X reduces to Y"

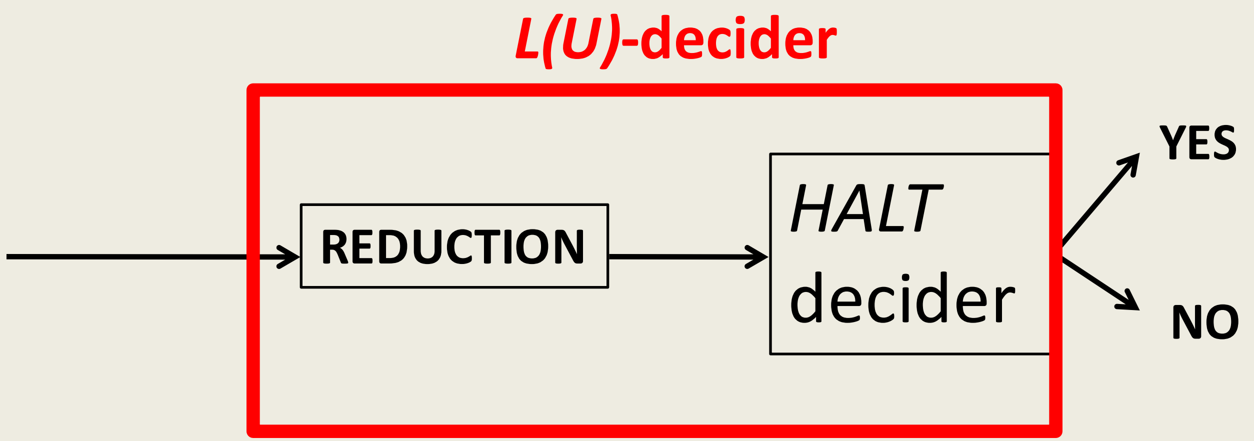


If Y can be decided, then X can be decided.

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

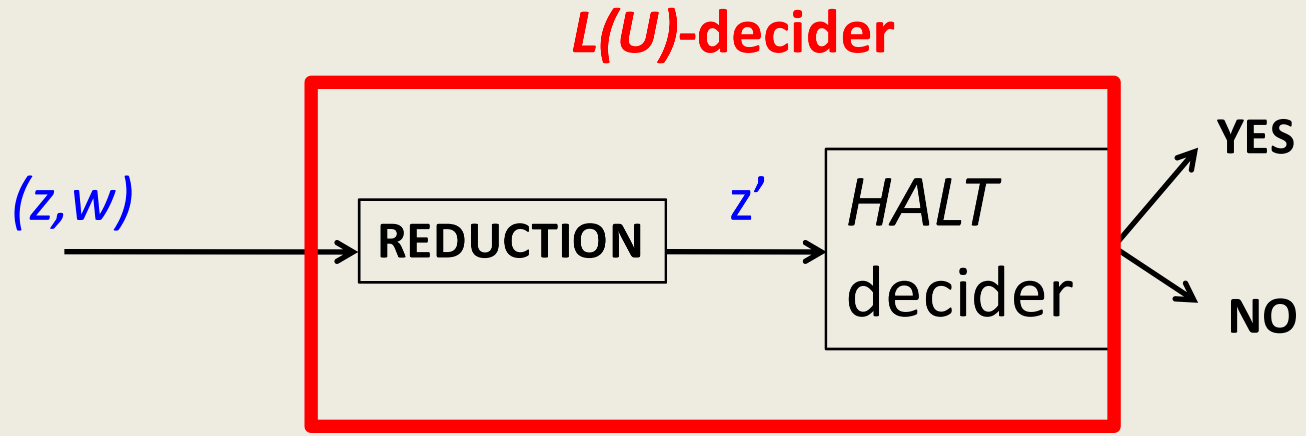
Halting Problem

- Does given *M* halt when run on blank input?
- *HALT* = { z | M_z halts when run on blank input}
- Show *HALT* is undecidable by showing
  L(U) ≤ *HALT*



What are input and output of the reduction?

$L(U) \leq HALT$



Need: $M_{z'}$ halts on blank input iff M_z accepts w

<u>TM</u> $M_{z'}$

const z

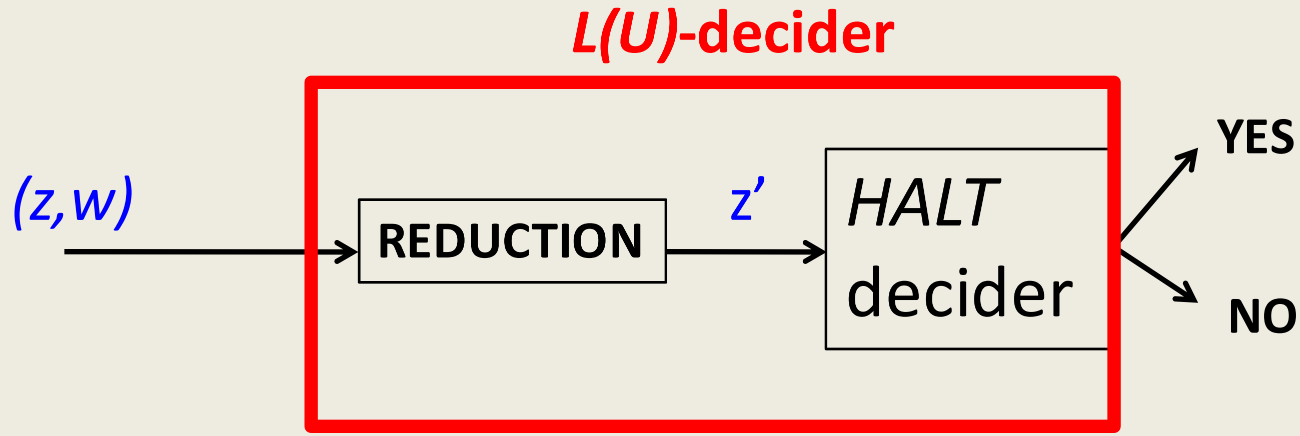
const w

run M_z on w and halt if it accepts

otherwise run for ever

The REDUCTION **doesn't run M_z on w**. It produces code for $M_{z'}$!

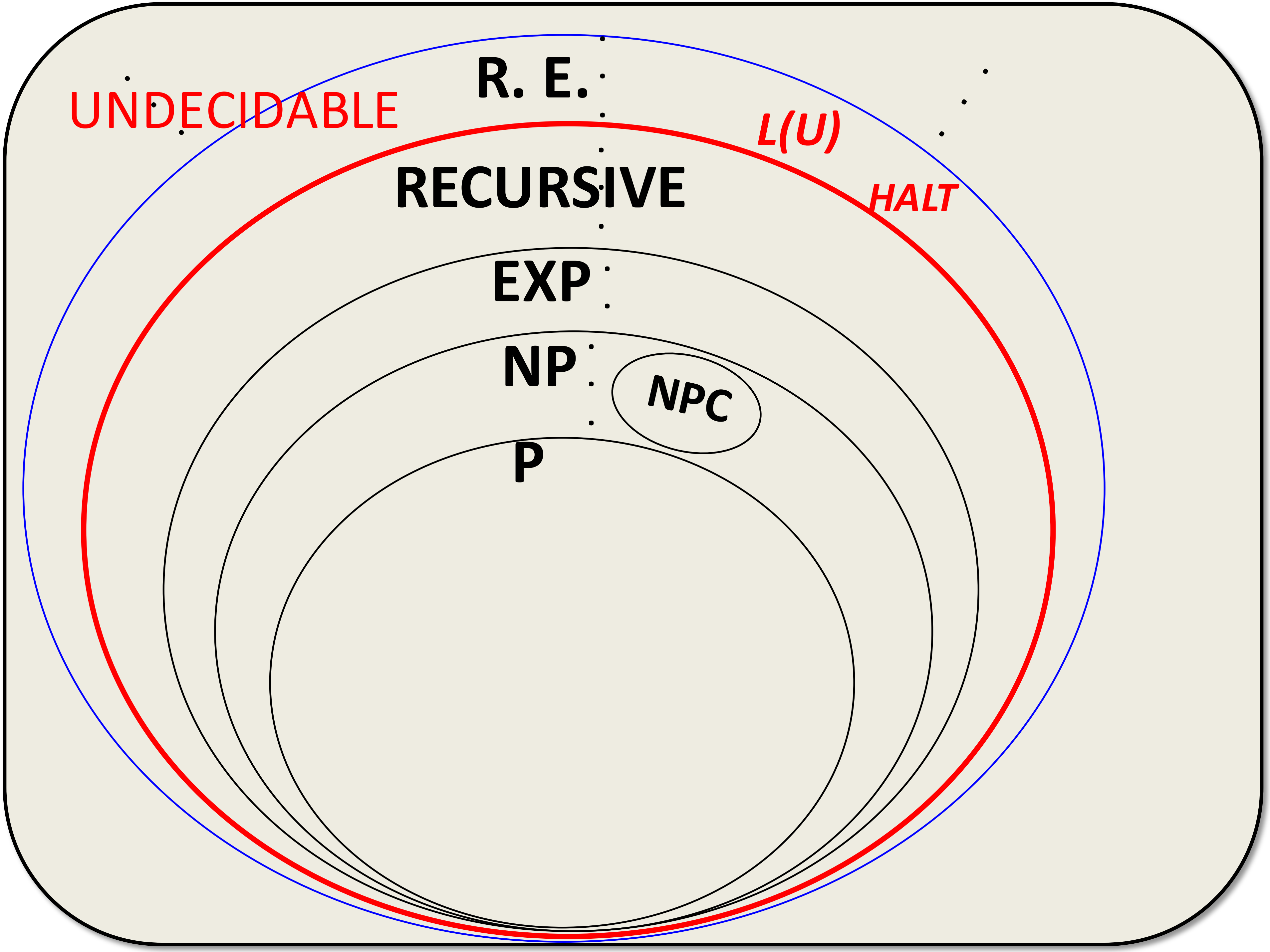
$L(U) \leq HALT$



Need: $M_{z'}$ halts on blank input iff M_z accepts w

TM $M_{z'}$
    const z
    const w
run M_z on w and halt if it accepts

Correctness: $L(U)$-decider say "yes" iff $M_{z'}$ halts on blank input
iff M_z accepts w
iff (z,w) is in $L(U)$



UNDECIDABLE
R. E.
L(U)
RECURSIVE
HALT
EXP
NP
NPC
P

Who cares about halting TMs?

- Remember, TMs = programs
- Virtually all math conjectures can be expressed as a halting-TM question.

Example: Goldbach's conjecture:

Every even number > 2 is the sum of two primes.

Program Goldbach

```
goldbach()
    n = 4
    WHILE is-sum-of-two-primes(n)
        n = n+2
    STOP AND SAY NO

is-sum-of-two-primes(n): boolean
    FOR p ≤ q < n
        IF p,q, prime AND p+q=n THEN RETURN TRUE
    RETURN FALSE
```

goldbach() halts iff Goldbach's conjecture is false

Deciding mathematical truth

prove-theorem(T)

```
w = " "
WHILE NOT is-a-proof-of (w,T)
    w = lexicographically-next-string(w)
OUTPUT T + "is true"
```

prove-theorem(T) halts iff there is a proof of T.

CS 125 assignment:

- Write a program that outputs “Hello world”.

```
main()
{   printf(“Hello world”);
}
```

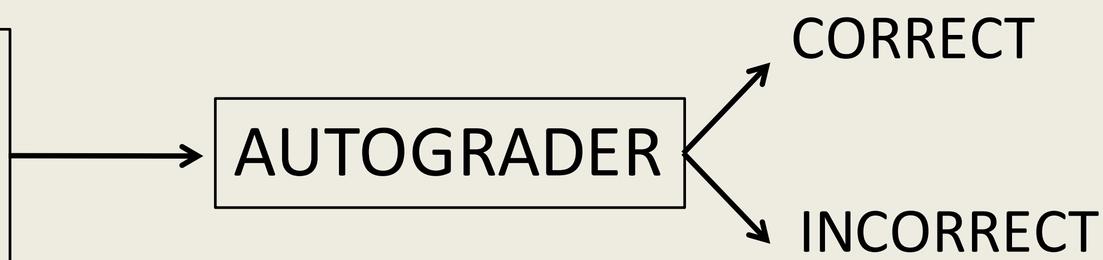
- Can we write an auto-grader?
- If so; we can solve Goldbach’s conjecture...

```
goldbach()
    n = 4
    WHILE is-sum-of-two-primes(n)
        n = n+2
    STOP AND SAY NO
```

```
is-sum-of-two-primes(n): boolean
    FOR p ≤ q < n
        IF p,q, prime AND p+q=n
            THEN RETURN TRUE
    RETURN FALSE
```

```
main()

  { goldbach();
   printf("Hello world");
  }
```



Deciding halting problem

- Given string *z*, to determine if program M_z halts, do the following:

So, deciding if a program prints "Hello world" is solving the halting problem

```
main()

   { Mz()
    printf("Hello world");
   }
```

→ AUTOGRADER → CORRECT / INCORRECT

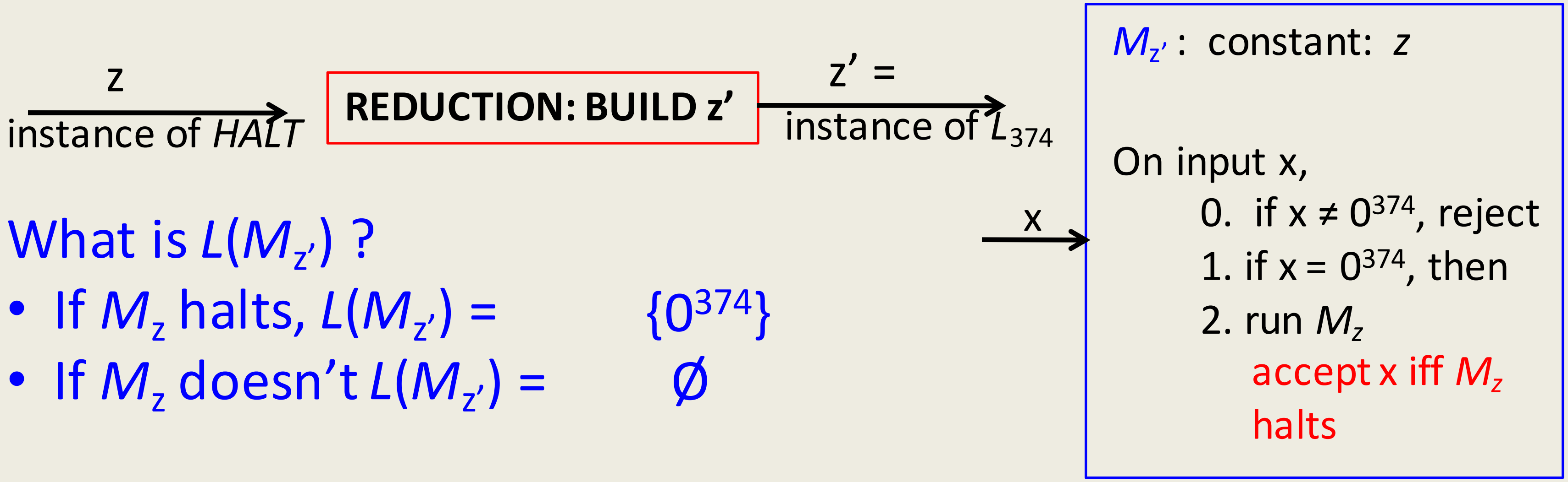
Using same ideas, we can show that deciding anything about code behavior is not possible

More reductions about languages

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

L_{374} = { z | $L(M_z)$ = {0^{374}} } is undecidable

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



What is $L(M_{z'})$?

- If M_z halts, $L(M_{z'})$ = {0^{374}}
- If M_z doesn't $L(M_{z'})$ = Ø

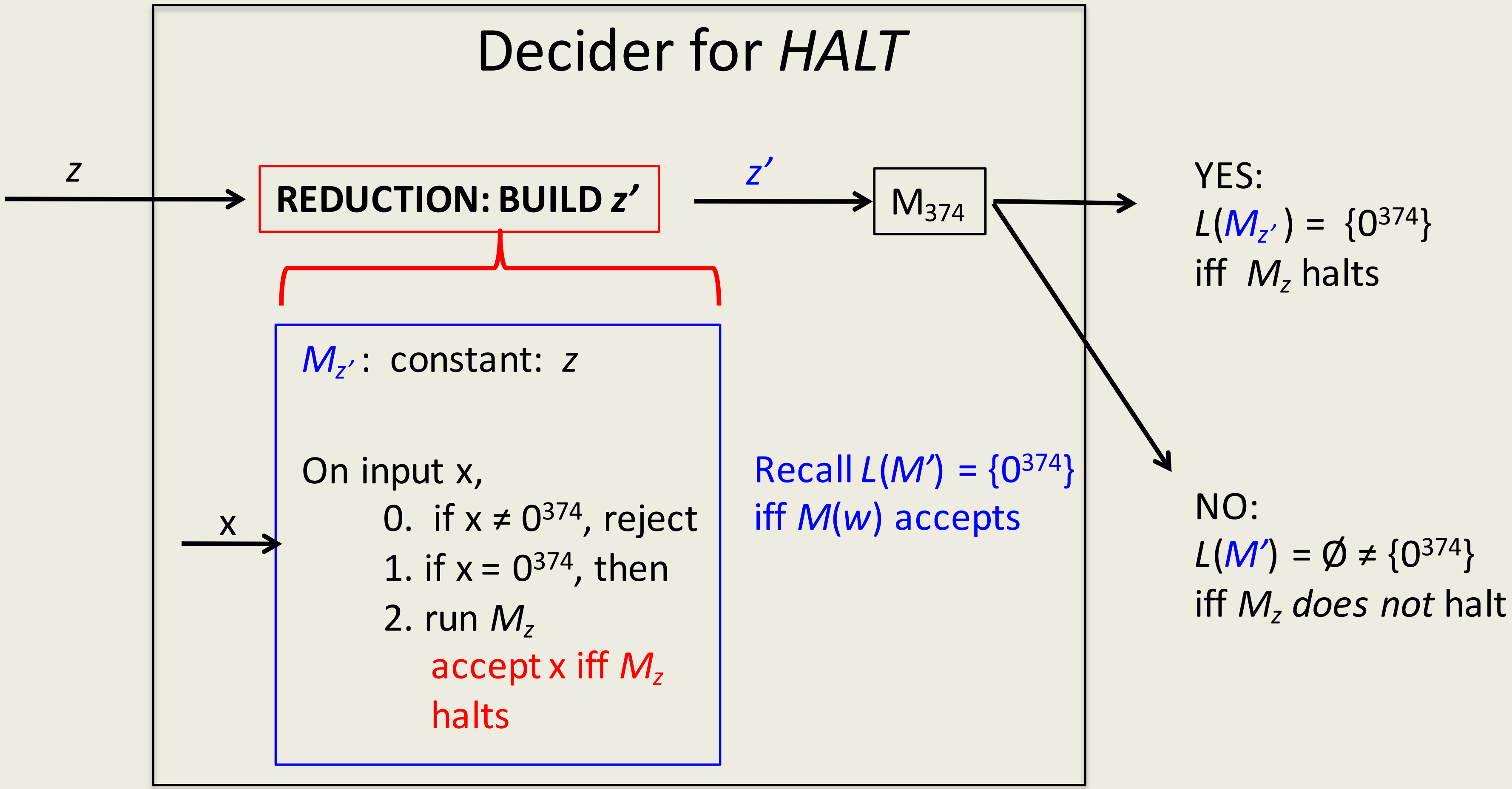
Q: How does the reduction know whether or not M_z halts ?

A: It doesn't have to. It just *builds* (code for) $M_{z'}$

```
main ( )
  {
    read input x
    run M_z ( )
    if (x = 0^374)
        accept
    else
        reject
```

accept x

If there is a decider M_{374} to tell if a TM accepts the language $\{0^{374}\}$...



Since *HALT* is not decidable, M_{374} doesn't exist, and L_{374} 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 $HALT \leq L_{\neq\emptyset}$

z → instance of *HALT* → **REDUCTION: BUILD z'** → z' = instance of $L_{\neq\emptyset}$

$M_{z'}$: constant: *z*

On input x,
Run M_z
Accept x if M_z halts

We want $M_{z'}$ to satisfy:

- If M_z halts, $L(M_{z'}) \neq \emptyset$
- If M_z doesn't $L(M_{z'}) = \emptyset$

If M_z halts, $L(M_{z'}) = \Sigma^*$ hence $\neq \emptyset$

If M_z doesn't, $L(M_{z'}) = \emptyset$

L_{pal} = { z | $L(M_z)$ = palindromes} is undecidable

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

We want $M_{z'}$ to satisfy:

- If M_z halts, $L(M_{z'})$ = {palindromes}
- If M_z doesn't $L(M_{z'})$ ≠ {palindromes}

$M_{z'}$: constant: *z*

On input x,

Run M_z

Accept x if

M_z halts and

x is a palindrome

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 $L(M) = \{0^{374}$ [illegible]
- Given *M*, is $L(M)$ [illegible]
- Given *M*, d[illegible]ny prime?
- Giv[illegible]ntain *any word?*
- [illegible] $L(M)$ meet these formal specs?
- G[illegible], 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"
  - $P(L)$ = "$L = \{0^n1^n \mid n \geq 0\}$"

Properties of r.e. languages

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

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

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 $\{ z \mid L(M_z) \text{ satisfies predicate } P\}$
- Note that each property P is thus a set of strings $L(P) = \{ z \mid L(M_z) \text{ satisfies predicate } P\}$
- **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.
- { z | $L(M_z)$ is r.e}.... why is this "trivial" ?
  - EVERY language accepted by an *M* is r.e. by def'n
- { z | $L(M_z)$ is not r.e}.... why is this "trivial" ?
- { z| $L(M_z) = \emptyset$ or $L(M_z) \neq \emptyset$}.... 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

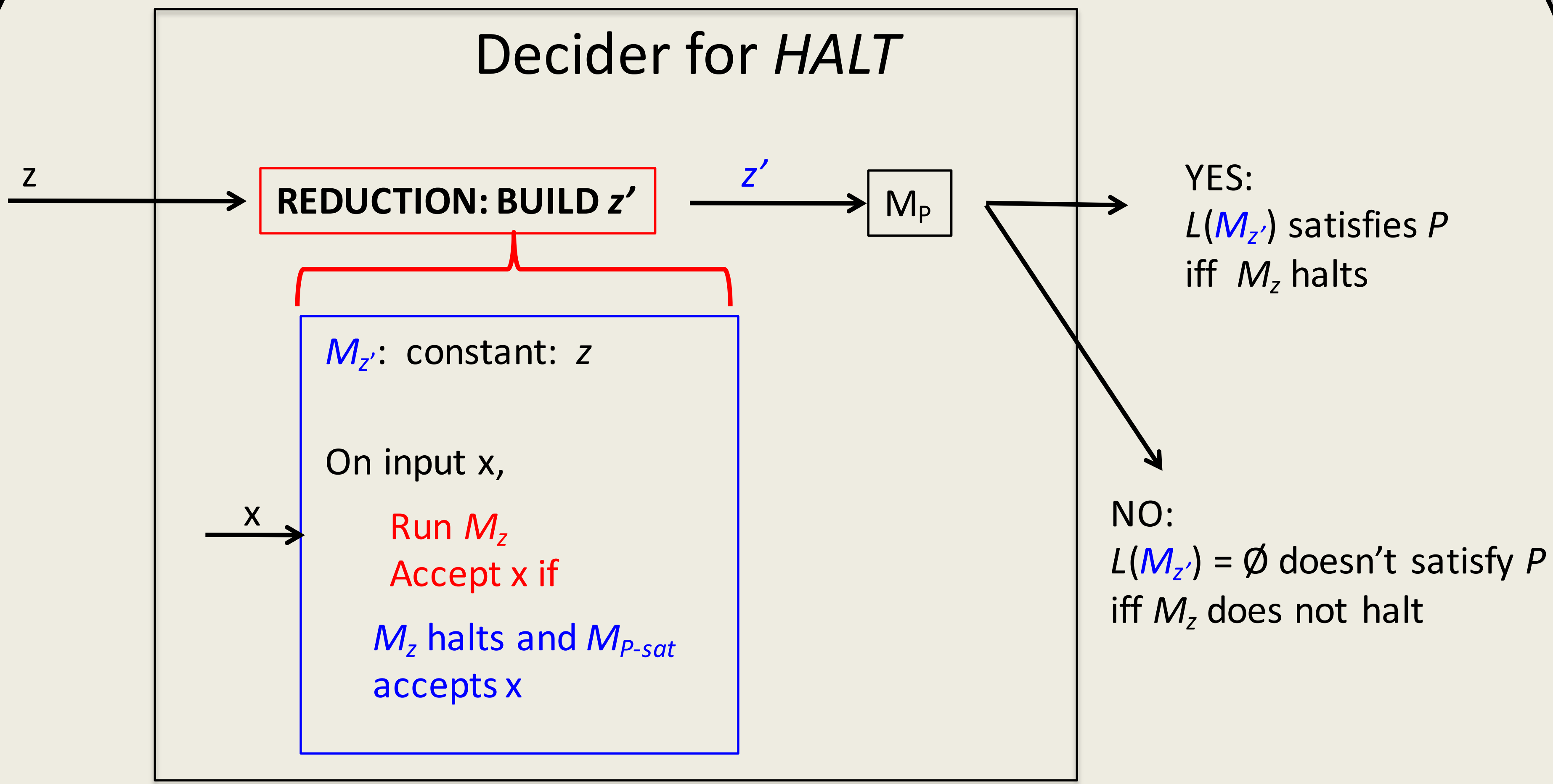
Example: auto-graders don't exist (if submissions are allowed to run an arbitrary (but finite) amount of time).

Proof

- Let *P* be a non-trivial property
- Let $L(P) = \{ z \mid L(M_z)$ satisfies predicate $P\}$
- Show *L(P)* is undecidable
- Assume Ø does not satisfy *P*
- Assume $L(M_{P\text{-}sat})$ satisfies *P* for some TM $M_{P\text{-}sat}$

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

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



If M_z doesn't halt then $L(M_{z'}) = \emptyset$

If M_z does halt then $L(M_{z'}) = L(M_{P\text{-}sat})$

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:
  - { z| M_z has 374 states}
  - { z| M_z uses ≤ 374 tape cells on blank input}
    - $374 \times |\Gamma|^{32} \times |Q_M|$
  - { z| M_z never moves head to left}
- sometimes undecidable
  - { z| M_z halts on blank input}
  - { z| M_z , on input "0110", eventually writes "2"}

Today

- Quick recap – halting & undecidability
- Undecidability via reductions
- Rice's theorem
- ICES
  - pick up TWO forms (Chandra + Manoj)
  - return to same location

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!

Final Thoughts

Grades are important but only in short term

No one will ask you how well you did in CS 374 in a year or two

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

Other Theory Courses

- “new” 473 (Theory 2) Jeff in Spring’16, Chandra in Fall’16
- Approximation algorithms (Chandra Spring’16)
- Computational Complexity (Kolla, Spring’16)
- Algorithmic Game Theory (Mehta, Spring ‘16)
- Randomized algorithms, Data structures, Computational Geometry, Algorithms for Big Data …

Other "Theory ish" Courses

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

Thanks!