CS 3313
Foundations of Computing:

Closure Properties of RE & Recursive Languages

Undecidable Problems

http://gw-cs3313.github.io

1

Decidable/Solvable Problems

- A problem is *decidable (solvable)* if there is an algorithm to answer it.
  - Recall: An "algorithm," formally, is a TM that halts on all inputs, accepted or not.
  - Put another way, *"decidable problem" = "recursive language."*
- Otherwise, the problem is *undecidable (unsolvable)*.

2

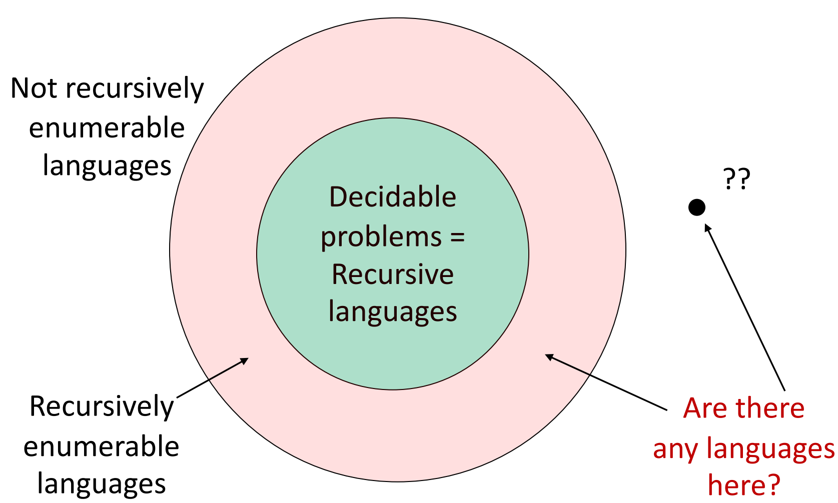
2

Closure Properties of Recursive and RE Languages

- Next topic is Decidability
  - Review Lab notes on Math review – diagonalization etc.
- First look at closure properties of these classes of languages
- Both are closed under union, concatenation, star, reversal, intersection, inverse homomorphism.
- Recursive closed under difference, complementation.
- RE closed under homomorphism.

3

Next….the BIG topic- Undecidable Problems
Bullseye Picture: Our immediate goal



4

Our first undecidable problem……

Recall: Enumeration of strings and Encoding TMs

- Canonical ordering of strings defines the *j-th* string w_j
  - Ordering: first by length and then for same length by 'numeric value'
    - 0, 1, 00, 01,….001,….,1100,…
- A Turing machine M can be encoded as a binary string <*M*>
  - $111\ldots110^i10^j10^k10^l10^m11\ldots111$
- Definition: Given an integer *i*, let M_j be the Turing machine whose code is binary representation of *i* denoted <*i*>
  - Many integers will not correspond to a TM code,
  - If <*i*> is not a valid TM code, then we take M_i to be a TM with one state and no transitions – i.e., M_i accepts empty set.
- Key takeaway: we can talk of the *j-th* string w_j and the *i-th* Turing machine M_i

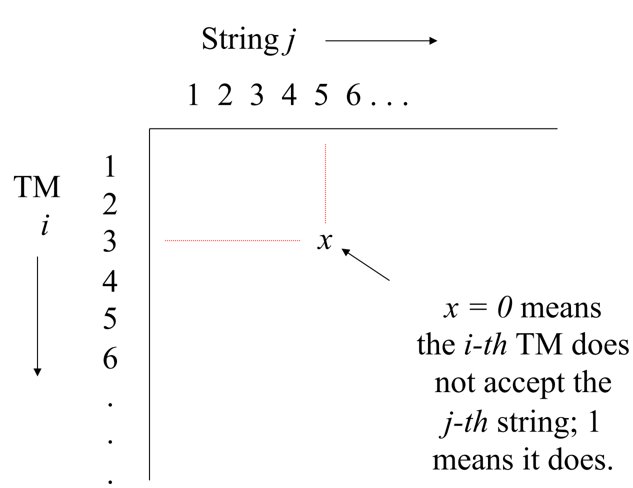
Table of Machines accepting Strings

- Analogous to lab discussion (on diagonalization – number of languages are not countable)…..
- Construct an infinite table with entries (i, j) to denote whether TM M_i accepts string w_j

  $(i, j) = 1$ if $w_j \in L(M_i)$ (w_j is accepted by M_i)

  $(i, j) = 0$ otherwise (w_j is not accepted by M_i)

Table of Acceptance



Diagonalization Argument and an undecidable language– (1)

- Consider the diagonalization language:

  $L_d = \{w \mid w$ is the *i-th* string, and the *i-th* TM does not accept $w\}$.

  - w_i is not accepted by M_i
- Diagonalization argument shows that L_d is not a recursively enumerable language; i.e., it has no TM !!!

9

9

L_d is undecidable (i.e., it is not recursive)– (2)

- Proof (by contradiction)
- *Assume L_d is a recursive language* (decidable problem)
- => there is a TM M^* that always halts that accepts this language.
- Every TM can be encoded as a binary string,

  therefore $M^* = M_j$ for some j.
- Consider string w_j : w_j is in $L(M_j)$ or w_j is not in $L(M_j)$

10

10

L_d is undecidable (L_d is not recursive)

- Assume L_d is a recursive language (decidable problem)
- => there is a TM M^* that always halts that accepts this language.
- Every TM can be encoded as a binary string, therefore $M^* = M_j$ for some j.
- Consider string w_j : w_j is in L(M_j) or w_j is not in L(M_j)
- *1. If* w_j is in $L(M_j)$ then entry $(j,j)=0$ which contradicts $w_j \in L(M_j)$
- *2. If* w_j is not in $L(M_j)$ then entry (j,j)=1 which implies w_j is accepted by M_j, which contradicts w_j is not in $L(M_j)$
- *Since for any string* w_j, either w_j is in $L(M_j)$ or w_j is not in $L(M_j)$, (1) and (2) contradicts the assumption that L_d=L(M*) for some TM M*
- *Therefore L_d is not accepted by any TM, i.e, it is not r.e.*

11

11

From the Abstract to the Real

- While the fact that L_d is undecidable is interesting intellectually, it doesn't impact the real world directly….
  - What is the relevance of this language to a "real" problem ?
- We first develop some TM-related problems that are undecidable, but our goal is to use the theory to show some real problems are undecidable.

12

12

Examples: Undecidable Problems

- Does a program halt on all inputs ?
  - Can we build a compiler that checks this.
- Can a particular line of code in a program ever be executed?
  - Version of question in Homework 8 !
- Is a given context-free grammar ambiguous?
- Do two given CFG's generate the same language?
- Is the language a regular (context free) language ?
  - Imagine how much better life in this course would be without the Pumping lemma!!

13

13

Another key proof technique: Reducability

- Remember key technique of Diagonalization ….old hat ☺
- Reducibility of a problem A to problem B
- Given two problems A and B,

  problem A is reducible to problem B if an algorithm for solving B can be used to solve problem A

  – *Therefore, solving A cannot be harder than solving B*

  - *If A is undecidable and A is reducible to B, then B is undecidable*
- Idea: If you had a black box that can solve instances of B, can you solve instances of A using calls to this Black box.
  - The black box is the assumed Algorithm for B.

14

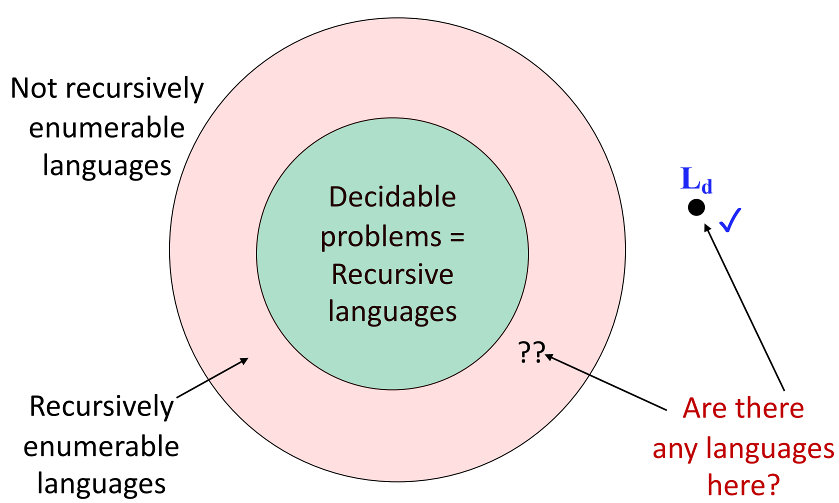
Reducibility: Why ?

- *Why is it useful*: Find one undecidable problem A, and then to show B is undecidable we construct a solution for A if we assume B is solvable/decidable.

- In context of time complexity: if we can reduce problem A to B in polynomial time and A is NP-complete then B is NP-complete.
  - To show a problem B is NP-complete, start with a NPC problem A (such as SAT) and show a polynomial time reduction to B

15

Undecidable Problems – the Bullseye Picture



16

Reducibility….Our first example

- An example of a recursively enumerable, but not recursive language is the language L_u of a *universal Turing machine UTM*.
- the UTM takes as input the code for some TM M and some binary string w and accepts *if and only if* M accepts w.
- Question: Given any arbitrary Turing machine M, does M halt on input w ?
- $L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$
- We designed a TM (the UTM) for L_u, so it is surely RE.
- *But is it recursive ? (decidable)*

17

Our current "collection" of undecidable languages

1. We proved that L_d *is not decidable (it is not even r.e.)*
   - $L_d = \{w \mid w = w_i \text{ and } M_i \text{ does not accept } w_i\}$.
2. If L_d is not recursive then its complement $\overline{L_d}$ is not recursive, i.e, it is undecidable

   $\overline{L_d} = \{w \mid w = w_i \text{ and } M_i \text{ accepts } w_i\}$.

18

18

L_u is Recursively Enumerable, but not Recursive

- We designed a TM (the UTM) for L_u, so it is surely RE.
- Assume it is recursive, i.e., there is a TM *M'* that always halts on input <*M,w*>.
- We now reduce $\overline{L_d}$ to L_u, i.e., if we had an algorithm for L_u then we could also design an algorithm for $\overline{L_d}$: contradiction.

  if L_u is decidable (solvable) then $\overline{L_d}$ is decidable

  $\overline{L_d}$ is undecidable

  example of $P \rightarrow Q$ and Not Q therefore Not P

19

L_u is Recursively Enumerable, but not Recursive

- To reduce L_u to $\overline{L_d}$:

1. Design an 'algorithm' *R:*

   a TM that takes input *w* and generates $<M_i, w_i>$

1. Feed $<M_i, w_i>$ to *M'* :

   M' accepts/halts $<M_i, w_i>$ if and only if *w* is in $\overline{L_d}$

   -- *contradiction.*

20

Do you remember this algorithm ?

- Generating sequences of strings/enumeration
- How to generate (i.e., enumerate) strings (numbers?) of increasing lengths from alphabet {0,1} ?
  - 0, 1, 00, 01, 10, 11, 000, ….
- Strings of increasing length of radix k, alphabet $\{1,1,\ldots k\}$
  - In our case radix $k = 2$ – the alphabet $\{0,1\}$
- Define*:* w_i is set of strings of length j
  - `w0 = { empty string }`
  - `for each string wj of length j  /* j=1 to n. */`

```
    for (radix k=2) x=0 to 1
            print  x.wj   /* concatenation */
```

21

L_u is Recursively Enumerable, but not Recursive

- We have an algorithm that generates sequence (canonical ordering) of strings $w_1, w_2, \ldots. w_i$
- Question 1: Can you design an algorithm that takes input w and determines value i such that $w = w_i$
- Question 2: Given an integer i, do you have an algorithm to represent i in binary ?
- Therefore:

22

22

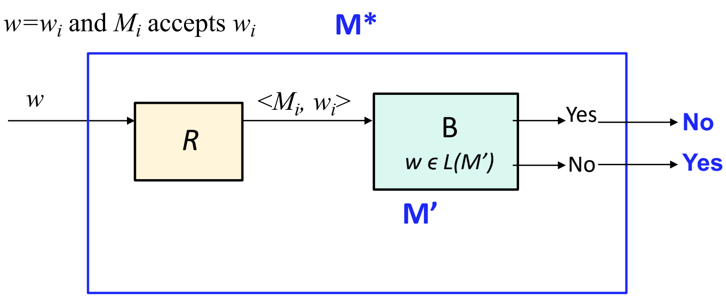
Proof: L_u is not Recursive: Proof – construct Algo R

- algorithm R (the reduction): Input is w and output is $<M_i, w_i>$

1. Use the canonical ordering algorithm to find i, where $w = w_i$
2. Generate binary representation of i, this is the code for M_i
3. Concatenate code for M_i and w_i to generate $<M_i, w_i>$

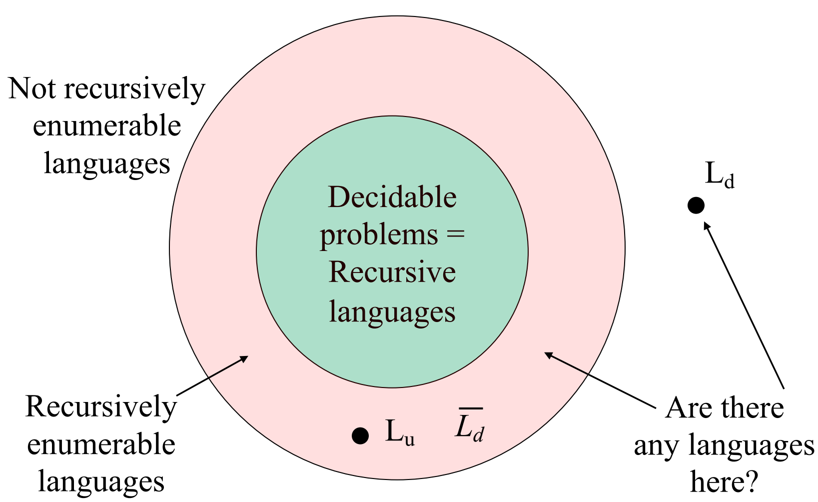
- *Send to hypothetical algorithm B for Halting Problem*
  - *B accepts if and only if w is in* $\overline{L_d}$

$w \in L(M^*)$ iff $w = w_i$ and M_i accepts w_i



23

Bullseye Picture



24

24

Halting Problem ? Other related problems….

- Does M halt on *w* ? = is L_u decidable
  - Original statement of the halting problem was slightly different but shown to be equivalent.
- Can we check if a program halts on all inputs = Does M halt on all inputs ?
- Is L(M) empty ?
- Is $L(M_1) = L(M_2)$ – i.e., are two programs equivalent ?
  - *Is this what an autograder program does ?*
- Can we check if a program enters a 'checkpoint' = Does M enter a state *q* ?
  - Variation of homework 8 question.

25

Summary…..

- Properties of Recursive and RE languages
- Concept of Decidability….
- Example of an Undecidable Problem
- Reducibility – prove other problems are undecidable
- Next: More examples of undecidable problems
  - Will post video of a reducibility example
  - Read the examples and exercises in the textbook
- and (last result in course)…Rice's Theorem: a powerful result that can be used to show (easily) that many properties of RE languages are undecidable.

26