Decidability

Turing Machines Coded as Binary Strings

Diagonalizing over Turing Machines

Problems as Languages

Undecidable Problems

Binary-Strings from TM's

- We shall restrict ourselves to TM's with input alphabet {0, 1}.
- Assign positive integers to the three classes of elements involved in moves:
  1. States: q_1(start state), q_2 (final state), q_3, ...
  2. Symbols X_1 (0), X_2 (1), X_3 (blank), X_4, ...
  3. Directions D_1 (L) and D_2 (R).

Binary Strings from TM's – (2)

- Suppose $\delta(q_i, X_j) = (q_k, X_l, D_m)$.
- Represent this rule by string $0^i10^j10^k10^l10^m$.
- Key point: since integers i, j, ... are all > 0, there cannot be two consecutive 1's in these strings.

Binary Strings from TM's – (2)

- Represent a TM by concatenating the codes for each of its moves, separated by 11 as punctuation.
  - That is: $\text{Code}_1 11 \text{Code}_2 11 \text{Code}_3 11$ ...

Enumerating TM's and Binary Strings

- Recall we can convert binary strings to integers by prepending a 1 and treating the resulting string as a base-2 integer.
- Thus, it makes sense to talk about "the i-th binary string" and about "the i-th Turing machine."
- Note: if i makes no sense as a TM, assume the i-th TM accepts nothing.

Table of Acceptance

String j

1 2 3 4 5 6 . . .

TM i

1
2
3
4
5
6
.
.
.

x

x = 0 means the i-th TM does not accept the j-th string; 1 means it does.

Diagonalization Again

- Whenever we have a table like the one on the previous slide, we can *diagonalize* it.
  - That is, construct a sequence D by complementing each bit along the major diagonal.
- Formally, $D = a_1a_2\ldots$, where $a_i = 0$ if the (i, i) table entry is 1, and vice-versa.

The Diagonalization Argument

- Could D be a row (representing the language accepted by a TM) of the table?
- Suppose it were the j-th row.
- But D disagrees with the j-th row at the j-th column.
- Thus D is not a row.

Diagonalization – (2)

- Consider the diagonalization language $L_d = \{w \mid w$ is the i-th string, and the i-th TM does not accept $w\}$.
- We have shown that L_d is not a recursively enumerable language; i.e., it has no TM.

Problems

- Informally, a "problem" is a yes/no question about an infinite set of possible *instances*.
- Example: "Does graph G have a *Hamilton cycle* (cycle that touches each node exactly once)?
  - Each undirected graph is an instance of the "Hamilton-cycle problem."

Problems – (2)

- Formally, a problem is a language.
- Each string encodes some instance.
- The string is in the language if and only if the answer to this instance of the problem is "yes."

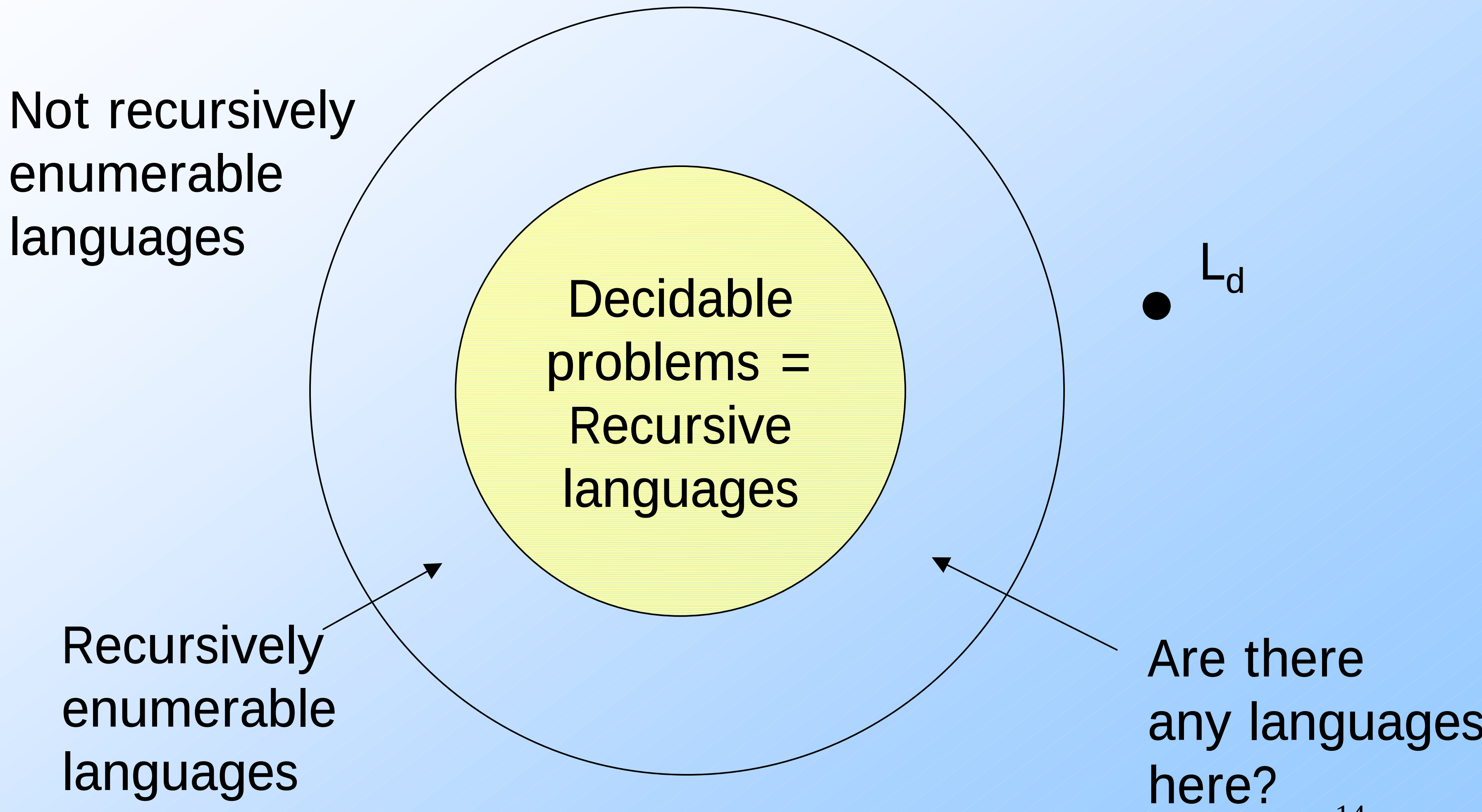
Example: A Problem About Turing Machines

- We can think of the language L_d as a problem.
- "Does this TM not accept its own code?"
- Aside: We could also think of it as a problem about binary strings.
  - Do you see how to phrase it?

Decidable Problems

- A problem is *decidable* 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*.

Bullseye Picture



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.
- We first shall develop some TM-related problems that are undecidable, but our goal is to use the theory to show some real problems are undecidable.

Examples: Undecidable Problems

- Can a particular line of code in a program ever be executed?
- Is a given context-free grammar ambiguous?
- Do two given CFG's generate the same language?

The Universal Language

- An example of a recursively enumerable, but not recursive language is the language L_u of a *universal Turing machine*.
- That is, 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.

Designing the UTM

- Inputs are of the form:

  Code for M 111 w

- Note: A valid TM code never has 111, so we can split M from w.
- The UTM must accept its input if and only if M is a valid TM code and that TM accepts w.

The UTM – (2)

- The UTM will have several tapes.
- Tape 1 holds the input M111w
- Tape 2 holds the tape of M.
  - Mark the current head position of M.
- Tape 3 holds the state of M.

The UTM – (3)

- Step 1: The UTM checks that M is a valid code for a TM.
  - E.g., all moves have five components, no two moves have the same state/symbol as first two components.
- If M is not valid, its language is empty, so the UTM immediately halts without accepting.

The UTM – (4)

- Step 2: The UTM examines M to see how many of its own tape squares it needs to represent one symbol of M.
- Step 3: Initialize Tape 2 to represent the tape of M with input w, and initialize Tape 3 to hold the start state.

The UTM – (5)

- Step 4: Simulate M.
  - Look for a move on Tape 1 that matches the state on Tape 3 and the tape symbol under the head on Tape 2.
  - If found, change the symbol and move the head marker on Tape 2 and change the State on Tape 3.
  - If M accepts, the UTM also accepts.

A Question

- Do we see anything like universal Turing machines in real life?

Proof That L_u is Recursively Enumerable, but not Recursive

- We designed a TM for L_u, so it is surely RE.
- Suppose it were recursive; that is, we could design a UTM U that always halted.
- Then we could also design an algorithm for L_d, as follows.

Proof – (2)

- Given input w, we can decide if it is in L_d by the following steps.

1. Check that w is a valid TM code.
   - If not, then its language is empty, so w is in L_d.
2. If valid, use the hypothetical algorithm to decide whether w111w is in L_u.
3. If so, then w is not in L_d; else it is.

Proof – (3)

- But we already know there is no algorithm for L_d.
- Thus, our assumption that there was an algorithm for L_u is wrong.
- L_u is RE, but not recursive.

Bullseye Picture