1 Introduction to Undecidability

1.1 Recap and Overview

- Continuing from the previous lecture on Friday, the focus is on understanding the limits of computation.
- We are exploring the concept of a **Universal Turing Machine** and its implications.
- We're going to define the **Diagonalization Language** (LD).
- We will also discuss the **Halting Problem**, which is related to the concepts discussed here.

1.2 Encoding Turing Machines

- To feed a Turing machine to itself, we need to **encode** the machine.
- Turing machines are encoded as a series of zeros and ones.
- Transitions are encoded using a sequence of zeros, with ones used as separators.

1.3 Encoding Details

- States are represented by a number of zeros (e.g., state 1 = '0', state 2 = '00', etc.).
- The tape symbol is also encoded with a number of zeros, separated by a one from the state encoding.
- The next state is a number of zeros representing the target state.
- The symbol to be written is encoded with zeros and separated with a one.
- The direction of the tape head movement (right or left) is encoded with a '1' '0' for right and '2' '0''s for left.
- Transitions are separated by two ones ('11').
- The input is separated from the transitions by three ones ('111').
- An encoding of a Turing machine and an input 'W' can be viewed as a long string.

1.4 Multiple Encodings

- The same Turing machine can have multiple encodings due to different orderings of transitions.
- For a machine with four states, there are *24* possible encodings (factorial 4).

2 Diagonalization Language (LD)

2.1 Definition

- The **Diagonalization Language** (LD) consists of strings ‘WI‘ such that ‘WI‘ is *not* in the language of the Turing machine ‘MII‘, where ‘MII‘ is encoded as the string ‘WI‘.
- Basically, the language consists of all the strings such that, when given as an input to a machine M whose code is ‘W‘, it will not be accepted.

2.2 Table Representation

- We can enumerate encodings of Turing machines (M1, M2, M3...) and their inputs (W1, W2, W3...).
- Create a table T where cell *T(i, j)* is ‘1‘ if ‘Wj‘ is accepted by ‘Mi‘, and ‘0‘ otherwise.

Table: Example table

2.3 Diagonalization

- The diagonalization language is created by taking the diagonal of this table and flipping the bits.
- If ‘T(i, i)‘ is 0, then ‘Wi‘ is in LD, and if ‘T(i, i)‘ is 1, then ‘Wi‘ is not in LD.
- This language consists of strings ‘Wi‘ such that ‘Mi‘ does not accept ‘Wi‘.

2.4 Key Takeaway

- The diagonalization language is constructed to be the opposite of what the Turing machine would accept.
- LD is specifically created to be in opposition to the behavior of Turing machines on their own encodings.

3 LD is Not Recursively Enumerable

3.1 Proof by Contradiction

- We prove that LD is not recursively enumerable using proof by contradiction.
- We assume that LD *is* recursively enumerable.
- We show that this assumption leads to a contradiction.

3.2 Contradiction Argument

- If LD is recursively enumerable, there must be a Turing machine MK that accepts it.
- We ask the question: is string ‘WK‘ in the diagonalization language LD.

3.3 Case Analysis

- **Case 1**: If ‘WK‘ is in LD, it means ‘MK‘ must accept it. In the table this means there is a ‘1‘ in the entry ‘T(k, k)‘. But by the definition of LD, ‘WK‘ cannot be in LD because its complement. This is a contradiction.
- **Case 2**: If ‘WK‘ is not in LD, then it means that ‘MK‘ does not accept it. In the table this means ‘T(k, k)‘ = 0, but then ‘WK‘ should be in LD because it’s the complement. This is again a contradiction.

3.4 Conclusion

- Since both cases lead to a contradiction, our original assumption that LD is recursively enumerable is false.
- Thus, LD is not recursively enumerable.
- There is a language that no Turing Machine can accept (no algorithm exists).

4 Relationship to Other Language Classes

4.1 Language Classes

- **Recursive Languages**: Algorithms that always halt with a yes or no answer.
- **Recursively Enumerable Languages**: Procedures that can guarantee a yes, but not guarantee a no.
- **Beyond Computation**: Languages that cannot be processed by any Turing machine.

4.2 Significance

- LD is in the class of languages beyond computation, making it an example of something not computable.
- This demonstrates the limitations of Turing machines as abstract computing devices.

5 Universal Turing Machine (UTM)

5.1 Universal Turing Machine

- The Universal Turing Machine (UTM) is a Turing machine that can simulate any other Turing machine.
- It takes as input an encoding of a Turing machine ‘M‘ and an input string ‘W‘.
- It then simulates ‘M‘ on ‘W‘.

5.2 UTM Setup

- The UTM has three tapes:
  - Tape 1 stores the encoding of ‘M‘ and the input ‘W‘.
  - Tape 2 represents the tape of the machine ‘M‘ is simulated.
  - Tape 3 represents the instantaneous description of ‘M‘.
- The UTM has a finite state control that handles simulating ‘M‘.

5.3 UTM Operation

- The UTM checks if the input encoding is valid.
- It initializes Tape 2 with the input ‘W‘.
- It simulates ‘M‘ on ‘W‘ by processing transitions from Tape 1.
- The UTM accepts if the simulated machine ‘M‘ accepts.

5.4 Universal Language (LU)

- The language of UTM is called the universal language (LU).
- LU consists of all pairs ‘¡M, W¿‘ such that ‘M‘ accepts ‘W‘.

6 Undecidability of LU

6.1 LU is Recursively Enumerable

- LU is recursively enumerable because we designed a Turing machine (UTM) that recognizes it.

6.2 LU is Not Recursive

- We use a proof by contradiction to show that LU is not recursive.
- We assume LU is recursive, meaning a Turing machine can always halt giving a yes or no answer.

6.3 Proof by Reduction

- We use the diagonalization language (LD) to prove the undecidability of LU.
- We show that if LU was recursive we could decide if a string is in LD. Which is already proven to not be possible.
- We assume there's a Turing machine that always halts that decides if '¡M, W¿' is in LU.
- If we could do this, it would mean that we can construct an algorithm for the diagonalization language, which we already know is not possible.
- Therefore, LU is not recursive.

6.4 Conclusion

- LU is a recursively enumerable language that is not recursive (it is in the donut-shape).
- It lies in the class of undecidable problems.

7 Halting Problem

7.1 Definition

- The Halting Problem asks whether a given Turing machine M will halt (terminate) when given an input W.
- The language 'H' for the halting problem is the set of '¡M, W¿' such that 'M' halts on 'W'.

7.2 Formulation

- The halting problem can be phrased as: given a Turing machine 'M' and input string 'W', does 'M' halt on 'W'?
- This is a classic undecidable problem.

7.3 Intuition

- The halting problem is similar to the Hello World and Full Program problems.
- It is a non-trivial property of a language.
- Non-trivial properties of languages are undecidable.

7.4 Halting is Recursively Enumerable

- The halting language is recursively enumerable because we can simulate ‘M‘ on ‘W‘ with a UTM.
- If ‘M‘ halts, our UTM can modify its state to accept.

7.5 Halting Problem is Not Recursive

- We prove the Halting Problem is not recursive via contradiction.
- We assume that we can decide if a machine holds on a certain input.

7.6 Proof

- We create a new machine M’ such that if M halts, M’ goes into an infinite loop, and if M rejects then M’ accepts.
- This change can be done easily by switching the acceptance and rejection of the original Turing Machine ‘M‘ on input ‘W‘.
- So if ‘MH‘ accepts, we go to an infinite loop, and if ‘MH‘ rejects then we accept.
- ‘MH‘ on W halts if only if ‘M’‘ on ‘W‘ rejects.
- This leads to a contradiction.
- Therefore, the halting problem is undecidable.

7.7 Conclusion

- The Halting Problem is an example of a problem that no Turing machine can solve.
- The halting problem is undecidable.

8 Final Thoughts

8.1 Key Takeaways

- Diagonalization language LD, and the universal language LU, and the halting problem are all undecidable.
- The diagonalization language is crucial for proving the undecidability of both the language of the universal touring machine and the halting problem.
- The UTM, a Turing machine capable of simulating any Turing machine, is essential to the discussion of the limits of what we can compute.

8.2 Important note

- Understanding proofs is more important than memorizing them. Focus on the concepts and contradictions of the proofs.