Theory of Computation (CS C351)

BITS Pilani
Hyderabad Campus

Dr.R.Gururaj
CS&IS Dept.

Undecidability(Ch.5)

Introduction

We tried to address the question-

- What can be computed
- What cannot be computed

We have introduced various and diverse mathematical models of computational processes that accomplish complex tasks- particularly, decide, semidecide, or generate languages, and compute functions.

- Turing machines can carry out surprisingly complex tasks of this sort.
- We have also seen that the additional features added to the standard TM do not increase the set of tasks that can be accomplished.
- We arrived at a class of languages that can be accepted by TMs and generated by TMs.

All this suggests that we have reached a natural upper limit on what computational device can be designed .

Our search for ultimate mathematical model of computational process, of an algorithm has been concluded successfully.

Turing Machine is the right answer.

- Turing machines deserve to be called as algorithms.
- But not all TMs are algorithms, since TM s that semidecide a language, and thus reject by never halting are not useful computational devices.
- Turing machines that decide and compute functions are of great use and we call them as algorithms.

Hence our notion of an algorithms must exclude Turing machine that may no halt on some inputs.

Nothing will be considered as an algorithm if it cannot be rendered as a TM that is guaranteed to halt on all inputs, and all such machines will be rightfully called as algorithms.

This is known as *Church-Turing Thesis*.

It is proved that certain problems can not be solved by any algorithm.

According to the Church-Turing thesis, computational tasks that cannot be performed by TMs are impossible, hopeless, and *undecidable*.

Universal Turing Machines

- The Turing machines we have seen so far are unprogrammable piece of hardware, specialized to solve one particular problem, with instructions that are hard-wired. The notion of HW.

- We shall argue that Turing machines are also SW.

We shall so that there is a certain generic TM that can be programmed about the same way that a general purpose computer can, to solve problems that can be solved by TMs.

The “program” that makes this generic machine behave like a specific machine *M* will have to be a description of *M*.

We consider the formalism of TMs as a programming language, in which we can write programs.

- Programs written in this language can be interpreted by a universal TM – that is to say another program written in the same language.

U is a Universal TM

M is a specific TM

"M" is the description of M

"w" is the description of the input string *w*

U halts on input *"M" "w"* if and only if *M* halts on *w*

U("M" "w") = "M(w)"

How Universal TM works

The Universal Turing Machine *U'* We use three tapes-

Tape-1 contains the encoding of current tape content.

Tape-2 Contains the encoding of M

Tape-3 Contains the encoding state of M at the current point in the simulated computation.

How Universal TM works

The Universal Turing Machine *U'* is started with some string "M" "w" on its first tape. And other two tapes are blank.

First U' moves "M" onto the second tape and shifts "w" down to the left end of the first tape, preceding it by "ΔU".

Thus at this point first tape contains "ΔUw".

Now U' writes the encoding of the initial state onto the third tape.

First U' scans the second tape until it finds a quadruple whose first component matches the encoded state written in its third tape, and whose second component matches the encoded symbol in the first tape.

If it finds such a quadruple, it changes the state to the third component of the quadruple and performs in the first tape the action suggested by the forth component.

This will continue.

If at some step, the state-symbol combination is not found in the second tape means it is a halting state. Then U' also halts at an appropriate state.

Undecidable problems about Turing machines.

The class of recursive languages is strict subset of the class of recursively enumerable languages.

The class of recursively enumerable languages is not closed under complement.

Any algorithm can be turned into a Turing Machine that halts on all inputs.

There is no algorithm that decides, for an arbitrary given Turing Machine M and input string w, whether or not M accepts w.

Undecidable problems about Turing machines.

Problems for which no algorithm exists is called *undecidable* or *unsolvable*.

The most famous and fundamental unsolvable problem is the one of telling whether a given TM halts on a given input.

This problem is usually called the *halting problem* for Turing machines.

Undecidable problems about TMs

1. Given a *M* and input *w* , does *M* halts on input *w*?
2. Given M, does M halts on the empty tape?
3. Given M is there any string at all on which it halts
4. Given M, does M halt on every input
5. Given M1, and M2, do they halt on same input strings
6. Given M, is the language that M semidecides regular? Is it CF? Is it recursive?

Unsolvable problems about Grammars

1. Given a grammar *G*, and string *w*, to determine whether *w€L(G)*
2. Given a grammar *G*, to determine whether *e€L(G)*
3. For two grammars *G1* and *G2*, to determine whether *L(G1) = L(G2)*
4. For an arbitrary grammar *G*, to determine whether *L(G)=ϕ*

Unsolvable problems about CFGs

1. Given a CFG *G*, is L(G)=∑*
2. Given two PDAs M1 and M2 , do they accept precisely the same language.
3. *Given a PDA M, find an equivalent PDA with as few states as possible.*
4. For two CFGs *G1* and *G2*, to determine whether *L(G1) = L(G2)*
5. For an arbitrary grammar *G*, to determine whether *L(G)=ϕ*