SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

FOUNDATIONS OF COMPUTER SCIENCE
LECTURE 13: Undecidability

Prof. Daniele Gorla

Undecidable problems

- Computers appear so powerful that you may believe that they can solve all problems
- One of the philosophically most important theorems of the theory of computation: There are problems that are algorithmically unsolvable
- So, no matter how powerful a computer and how smart the programmer, today we shall prove that computers are limited in a fundamental way

- Even ordinary problems that people want to solve turn out to be computationally unsolvable
- For example: given a computer program and a precise specification of what that program is supposed to do, you need to verify that the program performs as specified
- Because both the program and the specification are mathematically precise objects, you hope to automate the process of verification by feeding these objects into a suitably programmed computer
- The general problem of software verification is not solvable by any computer

Membership problem for TMs (1)

Thm.: $A_{TM} = \{ \langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ is undecidable.

Proof

We assume that A_{TM} is decidable and obtain a contradiction.

Suppose that H is a decider for A_{TM}, i.e.

$$H(\langle M, w\rangle) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \\ \text{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

Next, we construct a new Turing machine D with H as a subroutine:

$$D(\langle M\rangle) = \begin{cases} \text{accept} & \text{if } H(\langle M, \langle M\rangle\rangle) = \text{reject (i.e., if } M \text{ does not accept } \langle M\rangle) \\ \text{reject} & \text{if } H(\langle M, \langle M\rangle\rangle) = \text{accept (i.e., } M \text{ accepts } \langle M\rangle) \end{cases}$$

rem: the encoding of a TM is a string of symbols, often binary, so it can be used as a word

The contradiction arises when we run D on its own description $\langle D\rangle$:

$$\langle D\rangle \in L(D) \text{ if and only if } \langle D\rangle \notin L(D)$$

Thus, there cannot exist any decider H for A_{TM}.

Q.E.D.

Membership problem for TMs (2)

Pictorially, let's describe the problem of A_{TM} as a matrix, where rows are TMs, columns are the encoding of TMs, and elements of the matrix are {accept, reject, BLANK} (where BLANK stands for non-termination):

$$
\begin{array}{c|ccccc}
 & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \langle M_4\rangle & \cdots \\
\hline
M_1 & accept & & accept & & \\
M_2 & accept & accept & accept & accept & \\
M_3 & & reject & & & \cdots \\
M_4 & accept & accept & & reject & \\
\vdots & & & \vdots & &
\end{array}
$$

H turns every BLANK into reject:

$$
\begin{array}{c|ccccc}
 & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \langle M_4\rangle & \cdots \\
\hline
M_1 & accept & reject & accept & reject & \\
M_2 & accept & accept & accept & accept & \cdots \\
M_3 & reject & reject & reject & reject & \\
M_4 & accept & accept & reject & reject & \\
\vdots & & & \vdots & &
\end{array}
$$

D complements the values in the diagonal:

$$
\begin{array}{c|ccccc}
 & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \langle M_4\rangle & \cdots \\
\hline
M_1 & \underline{reject} & reject & accept & reject & \\
M_2 & accept & \underline{reject} & accept & accept & \cdots \\
M_3 & reject & reject & \underline{accept} & reject & \\
M_4 & accept & accept & reject & \underline{accept} & \\
\vdots & & & \vdots & & \ddots
\end{array}
$$

But since *D* is a TM itself, it is present both in the rows and in the columns:

$$
\begin{array}{c|ccccccc}
 & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \langle M_4\rangle & \cdots & \langle D\rangle & \cdots \\
\hline
M_1 & \underline{reject} & reject & accept & reject & & accept & \\
M_2 & accept & \underline{reject} & accept & accept & \cdots & accept & \cdots \\
M_3 & reject & reject & \underline{accept} & reject & & reject & \\
M_4 & accept & accept & reject & \underline{accept} & & accept & \\
\vdots & & & \vdots & & \ddots & & \\
D & reject & reject & accept & accept & & \underline{\ ?\ } & \\
\vdots & & & \vdots & & & & \ddots
\end{array}
$$

Membership problem for TMs (3)

SAPIENZA
Università di Roma
Dipartimento di Informatica

Thm.: $A_{TM} = \{ \langle M, w\rangle \mid M$ is a TM and M accepts $w\}$ is recursively enumerable.

Proof

The following Turing machine U recognizes A_{TM}: different, as U is not a decider, so it can loop

On input $\langle M, w\rangle$, where M is a TM and w is a string:

1. Simulate M on input w
2. If M enters its accept state, accept; if M enters its reject state, reject.

Q.E.D.

- U loops on input $\langle M, w\rangle$ if M loops on w

  → this is why U cannot decide A_{TM}
- If the algorithm had some way to determine that M was not halting on w, it could reject in this case (like H in the previous proof)

  → there is no algorithmic way to establish this (see later on)
- The Turing machine U is interesting in its own right and it is called ***universal TM*** (first proposed by Alan Turing in 1936)
- This machine is called universal because it is capable of simulating any other TM from the description of that machine.
- Played an important early role in developing stored-program computers

Beyond R.E. languages

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Are there languages that are not either R.E.? i.e. languages not generated by any TM

Thm.: L is decidable if and only if both L and $\overline{L}$ are R.E..

Proof

→ If L is decidable, there exists a decider M for it. Hence, L is R.E. (a decider is a TM) but also $\overline{L}$ is R.E. (it is accepted by the TM that behaves like M, but with q_{accept} and q_{reject} swapped).

← Let M_1 be a TM for L and M_2 for $\overline{L}$. Then, consider the TM M that

On input w:

1. Run in parallel M_1 and M_2 on input w
2. If M_1 accepts, accept; if M_2 accepts, reject.

Running the two machines in parallel means that M has two tapes: one for simulating M_1 and the other for simulating M_2. Then, M performs one step of each machine, and continues until one of them accepts (that eventually happens, since every w either belongs to L or to $\overline{L}$).

Q.E.D.

Now consider $\bar{A}_{TM} = \{ s \mid s$ is not the encoding of a TM and a string$\} \cup$
$\{ \langle M, w \rangle \mid M$ is a TM and M does not accept $w\}$

Cor.: $\bar{A}_{TM}$ is not R.E. .

Proof

If it was, A_{TM} would have been decidable. Q.E.D.

The Halting problem

Thm.: $H_{TM} = \{ \langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$ is undecidable.

Proof

By contradiction, assume the existence of decider R for H_{TM}.

We construct a decider S for A_{TM} as follows:

On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run R on input $\langle M, w\rangle$
2. If R rejects, reject
3. If R accepts, simulate M on w until it halts
4. If M has accepted, accept; otherwise reject.

Clearly, if R decides H_{TM}, then S decides A_{TM}.

Because A_{TM} is undecidable, H_{TM} also must be undecidable.

Q.E.D.

Proofs by Reductions

- The previous proof is an example of ***reduction***, a crucial method in this course
- Reductions are ways of (algorithmically) converting a problem A into another problem B in such a way that a solution for B can be used to solve A
- This is something common in everyday life:
  - Suppose that you want to find your way around a new city
  - This would be easy if you had a map.
  - Thus, you can reduce the problem of finding your way around the city (problem A) to the problem of obtaining a map of the city (problem B)
- Note that reducibility says nothing about solving A or B alone

  → it only states that A can be solved in the presence of a solution to B

- In computability, this can be used whenever A is reducible to B and B is decidable; in this case, also A is decidable

  → Equivalently (as we did before):

  if A is undecidable and reducible to B, then B is undecidable too
- In proving undecidability of the Halting problem, we used A = A_{TM} and B = H_{TM}.

Rice's Theorem

Thm.: Let P be a language consisting of TM descriptions such that

1. P is nontrivial (i.e., it contains some, but not all, TM descriptions); and
2. P is defined by some property of the TM's language.

Then, P is undecidable.

Proof

By contradiction, let R_P be a decider for P; we now show how to reduce A_{TM} to P.

Let $T_\emptyset$ be a TM that always rejects, so $L(T_\emptyset) = \emptyset$.

W.l.o.g., assume that $\langle T_\emptyset \rangle \notin P$ (if not, proceed with $\bar{P}$ instead of P, since a decider for P yields one for $\bar{P}$).

Because P is not trivial, there exists a TM T with $\langle T \rangle \in P$. Given M and w, consider the TM $S_{T,M,w}$:

On input x:
(i) Simulate M on w
(ii) If it halts and rejects, reject
(iii) If it accepts, simulate T on x. If it accepts, accept

If M accepts w,
the language of $S_{T,M,w}$ is the same as T's, otherwise as $T_\emptyset$'s:

$$L(S_{T,M,w}) = \begin{cases} L(T) & \text{if } M \text{ accepts } w \\ L(T_\emptyset) & \text{otherwise.} \end{cases}$$

We now show how to decide A_{TM} by using R_P's ability to distinguish between $T_\emptyset$ and T:

On input $\langle M, w \rangle$:
- Run R_P with input $\langle S_{T,M,w} \rangle$. If it accepts, accept; otherwise reject

Therefore, M accepts w iff $\langle S_{T,M,w} \rangle \in P$. Q.E.D.

Corollaries of Rice's Theorem

Cor.: $E_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \}$ is undecidable.

Cor.: $EQ_{\mathrm{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$ is undecidable.

Cor.: $FIN_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a finite language} \}$ is undecidable.

Cor.: $REG_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language} \}$ is undecidable.

Cor.: $CF_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a C.F. language} \}$ is undecidable.

Cor.: $CS_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a C.S. language} \}$ is undecidable.

Computation histories

Def.: Let M be a Turing machine and w an input string. A ***computation history*** for M on w is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

- C_1 is the start configuration of M on w, and
- each C_i yields C_{i+1} according to the transitions of M.

The history is said ***accepting/rejecting*** whenever C_k is an accepting/rejecting configuration.

Computation histories are finite sequences.

→ If M doesn't halt on w, no accepting nor rejecting history exists for M on w

Deterministic machines have at most one computation history on any given input;

Nondeterministic machines may have many computation histories on a single input, corresponding to the various computation branches

→ in the rest of this class, we shall only consider deterministic TMs

Undecidable problems for CSLs: Emptyness

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Thm.: $E_{LBA} = \{ \langle M \rangle \mid M \text{ is a LBA and } L(M) = \emptyset \}$ is undecidable.

Proof

Assume a decider R for E_{LBA} and construct a decider S for A_{TM} as follows:

On input $\langle M, w \rangle$, where M is a TM and w is a string:

(i) Construct a LBA B that accepts all and only the accepting computation histories for M on w

(ii) Run R on input $\langle B \rangle$

(iii) If R accepts (i.e., $\langle B \rangle \in E_{\text{LBA}}$), reject (i.e., $w \notin L(M)$); if R rejects, accept

For step (i), we proceed as follows:

- we assume that a history is a single string with the configurations separated by #
- On input x, first B breaks up x according to the delimiters
- Then B determines whether the C_i's satisfy the conditions of an accepting computation history:
  1. C_1 is the start configuration for M on w
  2. Each C_{i+1} legally follows from C_i
  3. C_k is an accepting configuration for M.
- 1 and 3 are very easy to check. For 2, B has to check that C_i and C_{i+1} are identical except for the positions under and adjacent to the head in C_i.
- These positions must be updated according to the transition function of M

→ checkable by zig-zagging between corresponding positions of C_i and C_{i+1}. **Q.E.D.**

Undecidable problems for CSLs: Equivalence

<u>*Thm.:*</u> $EQ_{LBA} = \{ \langle M_1, M_2 \rangle \mid M_1$ and M_2 are LBA and $L(M_1) = L(M_2)\}$ is undecidable.

Proof

Assume a decider R for EQ_{LBA} and construct a decider S for E_{LBA} as follows:

On input $\langle M \rangle$, where M is a LBA:

1. Run R on input $\langle M, N \rangle$, where N is a LBA that rejects all inputs
2. If R accepts, accept; if R rejects, reject.

<u>Q.E.D.</u>

Undecidable problems for CFLs: All Σ^*

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Thm.: $ALL_{\text{PDA}} = \{ \langle M \rangle \mid M \text{ is a PDA and } L(M) = \Sigma^* \}$ is undecidable.

Proof

Assume a decider R for ALL_{PDA} and construct a decider S for A_{TM} as follows:

On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct a PDA B that accepts all Σ^* if and only if M doesn't accept w
2. Run R on input $\langle B \rangle$
3. If R accepts, reject; if R rejects, accept

B should accept everything but one accepting history for M on w (provided that one exists, i.e. $w \in L(M)$).

B receives histories still as strings of config's separated by # (almost...) and non-deterministically:

- Checks whether the first configuration is not the starting one → if so, accepts
- Checks whether the last configuration is not an accepting one → if so, accepts
- Checks whether some C_i doesn't yield C_{i+1} according to the transitions of M → if so, accepts

For the last task, B has a non-det. branch (for all i) that: reads C_i, pushes it into the stack, and then compares it with C_{i+1} (still around the head position) by simultaneously reading C_{i+1} and popping C_i

→ but C_i is in the wrong order (the last char of C_i is at top of the stack after the push)

Hence, the input for B is $\#\underbrace{\longrightarrow}_{C_1}\#\underbrace{\longleftarrow}_{C_2^{\mathcal{R}}}\#\underbrace{\longrightarrow}_{C_3}\#\underbrace{\longleftarrow}_{C_4^{\mathcal{R}}}\#\ \cdots$

Q.E.D.

Undecidable problems for CFLs: Equivalence

<u>*Thm.:*</u> $EQ_{\text{PDA}} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are PDA and } L(M_1) = L(M_2)\}$ is undecidable.

Proof

Assume a decider R for EQ_{PDA} and construct a decider S for ALL_{PDA} as follows:

On input $\langle M \rangle$, where M is a PDA:

1. Run R on input $\langle M, N \rangle$, where N is a PDA that accepts all inputs
2. If R accepts, accept; if R rejects, reject.

<u>Q.E.D.</u>

Problems for Languages: Summing up

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

	Membership	***Emptyness***	***Equivalence***
Regular	DEC	DEC	DEC
C.F.	DEC	DEC	UNDEC
C.S.	DEC	UNDEC	UNDEC
R.E.	UNDEC	UNDEC	UNDEC