COMP30026 Models of Computation

Reducibility

Bach Le / Anna Kalenkova

Lecture Week 12. Part 1

Semester 2, 2021

In the last lecture we saw that

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

is undecidable.

Tute exercise T12.1 asks you to prove that it follows that

$$Halt_{TM} = \left\{ \langle M, w\rangle \;\middle|\; \begin{array}{l} M \text{ is a Turing machine and} \\ M \text{ halts when run on input } w \end{array} \right\}$$

is also undecidable.

At Least A_{TM} Is Recognisable

Note that

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

is Turing recognisable.

The reason is that it is possible to construct a universal Turing machine U which is able to simulate any Turing machine.

On input $\langle M, w\rangle$, U simulates M on input w.

If M enters its accept state, U accepts.

If M enters its reject state, U rejects.

If M never halts, neither does U.

Hilbert's Tenth Problem

Mathematicians have this proud history of inventing algorithms and posing algorithmic challenges.

Here, for example, is the famous tenth problem from a list of 23 posed by David Hilbert in 1900:

David Hilbert

> *Find an algorithm that determines whether a polynomial (in many variables but with integer coefficients) has an integral root.*

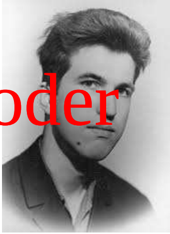
Hilbert's Tenth Problem

70 years after Hilbert posed his tenth problem, based on work by J. Robinson, Y. Matiyasevich proved that there is no algorithm for it.

Julia Robinson

$\{p \mid p$ is a polynomial with integral root$\}$ is undecidable.

However, this language does have a recogniser.



Y. Matiyasevich

If the input polynomial p has k variables $x_1, \ldots, x_k$ then we can simply enumerate all integer k-tuples $(v_1, \ldots, v_k)$ and evaluate $p(v_1, \ldots, v_k)$, one by one. If $p(v_1, \ldots, v_k) = 0$, accept. We refer to this type of problem as semi-decidable.

Closure Properties

The set of Turing recognisable languages is closed under the regular operations, and intersection. It is not closed under complement.

The set of decidable languages is closed under the same operations, and also under complement.

Week 11 tute exercises explore some of these closure results in more detail.

Relating Decidability and Recognisability

Theorem: A language L is decidable iff both L and its complement L^c are Turing recognisable.

Proof: If L is decidable, clearly L and also L^c are recognisable.

Assume both L and L^c are recognisable. That is, there are recognisers M_1 and M_2 for L and L^c, respectively.

A Turing machine M can then take input w and run M_1 and M_2 on w in parallel. If M_1 accepts, so does M. If M_2 accepts, M rejects.

Note that at least one of M_1 and M_2 is guaranteed to accept.

Hence M decides L.

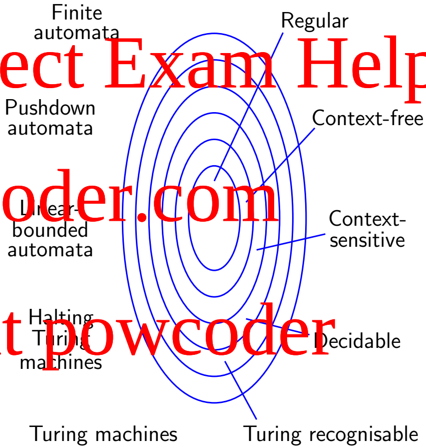
A Non-Turing Recognisable Language

This gives us an example of a language which is not Turing recognisable: $(A_{TM})^c$, that is, the complement of A_{TM}.

Namely, we know that A_{TM} is recognisable.

If $(A_{TM})^c$ was also Turing recognisable, A_{TM} would be decidable.

But we have shown that it isn't.



Too Many Languages

As we have seen, the set $\{0, 1\}^*$ of finite binary strings is a countable set.

Problem set exercise P12.3 asks you to show that the set $\mathcal{B}$ of all infinite binary strings is uncountable.

That is interesting, because it follows that the set of all languages over any finite non-empty alphabet Σ is also uncountable.

Namely, the set of all languages over Σ is in a one-to-one correspondence with $\mathcal{B}$, as we now show.

Let $s_1, s_2, s_3, \ldots$ be the standard enumeration of Σ^*.

For each language A over Σ, there is a unique characteristic sequence $\chi_A \in \mathcal{B}$, whose ith bit is 1 iff $s_i \in A$:

$$
\begin{array}{rcccccccccccc}
\Sigma^* : & \{ & \epsilon, & a, & b, & aa, & ab, & ba, & bb, & aaa, & aab, & \ldots & \} \\
A : & \{ & & a, & & aa, & ab, & & & aaa, & & \ldots & \} \\
\chi_A : & & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \ldots &
\end{array}
$$

Hence we cannot put the set of all languages into a one-to-one correspondence with the set of all Turing machines.

That is, we could never hope to have a recogniser for each possible language.

Reducibility

Let P and P' be decision problems with instances p_i and p'_i.

P can be reduced to P' iff there is a Turing machine M:

$$\langle p_i \rangle \longrightarrow \boxed{M} \longrightarrow \langle p'_i \rangle$$

such that $answer(p_i)$ can be obtained from $answer(p'_i)$.

- P reducible to P' and P' decidable $\Rightarrow$ P decidable.
- P reducible to P' and P undecidable $\Rightarrow$ P' undecidable.

So reducibility is useful for proving decidability and undecidability results.

TM Emptiness Is Undecidable

Theorem:

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

is undecidable.

Proof: A_{TM} is reducible to E_{TM}. First notice that, given $\langle M, w \rangle$, a Turing machine can modify the encoding of M, so as to turn M into M' which recognises $L(M) \cap \{w\}$.

Here is what the new machine M' does:

1. If input x is not w, reject.
2. Otherwise run M on w and accept if M does.

TM Emptiness Is Undecidable

Notice how w has been "hard-wired" into M': This machine is like M, but it has extra states added to perform the test against w.

Also note that

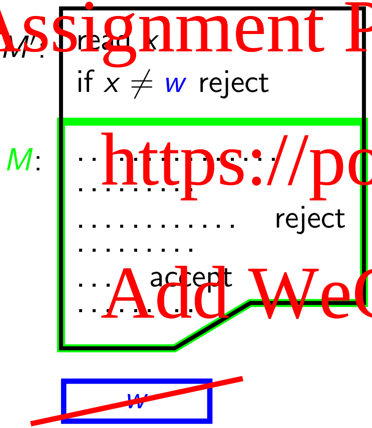
$$L(M') = \begin{cases} \{w\} & \text{if } M \text{ accepts } w \\ \emptyset & \text{otherwise} \end{cases}$$

Here then is a decider for A_{TM}, using a decider R for E_{TM}:

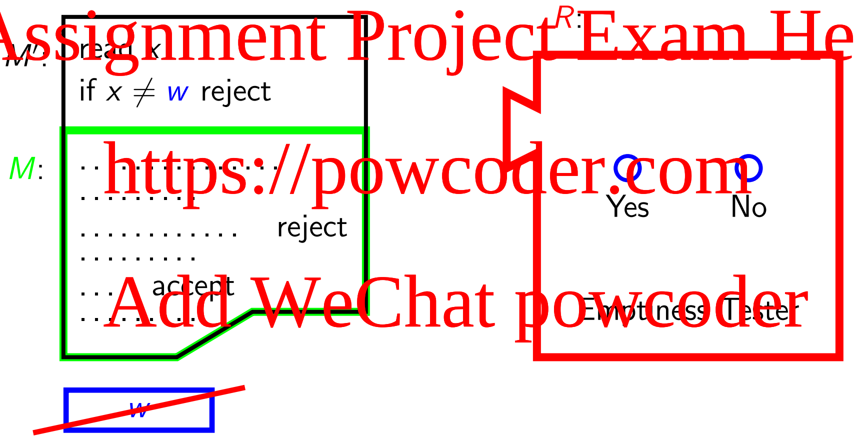
1. From input $\langle M, w\rangle$ construct $\langle M'\rangle$.
2. Run R on $\langle M'\rangle$.
3. If R rejects, accept; if it accepts, reject.

The Argument in Pictures

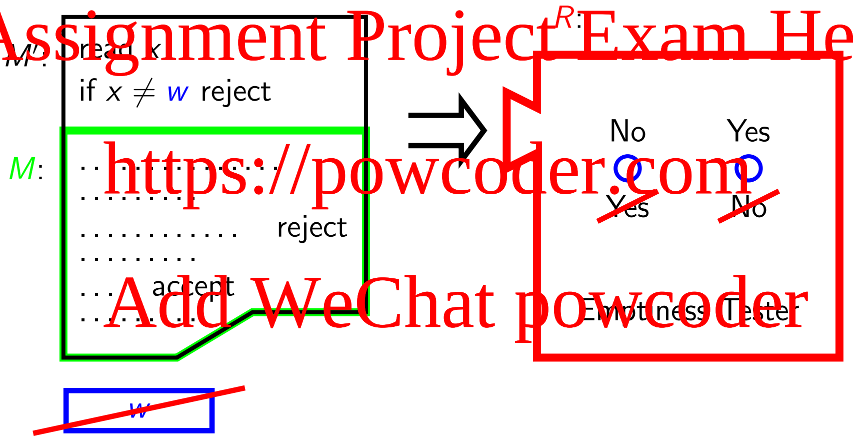
The Argument in Pictures



The Argument in Pictures



The Argument in Pictures



TM Equivalence Is Undecidable

Theorem:

$$EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

is undecidable.

Proof: E_{TM} is reducible to EQ_{TM}.

Assume that S decides EQ_{TM}. Here is a decider for E_{TM}:

1. Input is $\langle M \rangle$.
2. Construct a Turing machine $M_\emptyset$ which rejects all input.
3. Run S on $\langle \langle M \rangle, \langle M_\emptyset \rangle \rangle$.
4. If S accepts, accept; if it rejects, reject.

But we know that E_{TM} is undecidable. So EQ_{TM} is undecidable.

Valid and Invalid Computations

Recall how we captured a Turing machine configuration as a string (such as ba$q_5$bb).

A valid computation for Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ (on input w) is a string of form

$$C_1 \# C_2 \# \cdots \# C_k \#$$

where

1. C_1 is M's start configuration,
2. C_k is an accepting configuration
3. for each even $i \in \{2, \ldots, k\}$, $C_{i-1} \Rightarrow C_i^{\mathcal{R}}$
4. for each odd $i \in \{2, \ldots, k\}$, $(C_{i-1})^{\mathcal{R}} \Rightarrow C_i$

A string (over the same alphabet) which is not a valid computation is an invalid computation.

The Language of Invalid Computations

Rephrasing: A string w is an invalid computation iff one or more of the following properties hold:

0. w is not of the form $C_1\#\cdots\#C_k\#$ with C_i in $\Gamma^* Q \Gamma^*$.
1. C_1 is not a start configuration for M, that is, not in $q_0\Sigma^*$.
2. C_k is not accepting, that is, not in $\Gamma^* q_a \Gamma^*$.
3. $C_{i-1} \not\Rightarrow C_i^{\mathcal{R}}$ for some even i.
4. $(C_{i-1})^{\mathcal{R}} \not\Rightarrow C_i$ for some odd i.

The set of strings satisfying (0)–(2) are regular languages.

We claim the set of strings satisfying (3) is context-free (and so is the set satisfying (4)).

The Language of Invalid Computations

Here is how to build a PDA P for the set of strings for which $C_{i-1} \Rightarrow C_i^R$ is false for some even i.

1. P non-deterministically skips past an even number of $\#$ symbols.
2. While reading through C_{i-1}, P pushes onto its stack the symbols of the configuration C obtained by $C_{i-1} \Rightarrow C$.
3. After skipping the next $\#$, P compares C (backwards) against the string found until the following $\#$; if they are different, P scans over the remaining input and accepts.

Conclusion: The language of invalid computations is context-free!

CFG Exhaustiveness is Undecidable

Theorem:

$$ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$$

is undecidable.

Proof: We just saw that, for any Turing machine M, we can construct a CFG G that generates all the invalid computations for M.

Clearly $L(G) = \Sigma^*$ iff $L(M) = \emptyset$.

Hence if ALL_{CFG} is decidable, then so is E_{TM}.

Since we proved the latter undecidable, ALL_{CFG} must be undecidable as well.

CFG Equivalence is Undecidable

Our final language undecidability result is an easy reduction from ALL_{CFG}.

Theorem:

$$EQ_{CFG} = \{\langle G, G' \rangle \mid G, G' \text{ are CFGs and } L(G) = L(G')\}$$

is undecidable.

Proof: Assume we have a decider E for EQ_{CFG}. Then we can easily build a decider for ALL_{CFG}. Namely, given input grammar G over alphabet Σ, construct a CFG G_{all} for Σ^*. Then run E on $\langle G, G_{all} \rangle$.

Undecidability in Logic

Around 1930, first-order logic had been found to have a sound and complete axiomatisation (by Kurt Gödel).

What was then considered the foremost outstanding problem of mathematical logic was the so-called Entscheidungsproblem: whether there is a decision procedure for first-order logic.

> *The Entscheidungsproblem is solved when one knows a procedure by which one can decide in a finite number of operations whether a given logical expression is generally valid or is satisfiable. The solution [...] is of fundamental importance for the theory of all fields, the theores of which are all capable of logical development from finitely many axioms.*
>
> *David Hilbert and Wilhelm Ackermann, 1928*

Undecidability in Logic

As Hilbert and Ackermann point out, if the answer to the question "is there a decision procedure for first-order logic" was yes (as was commonly assumed), that would have hugely important ramifications.

It would mean that every theory that can be formalised in first-order logic would have an algorithm for deciding the truth of assertions in that theory.

Wilhelm Ackermann

Alan Turing set out to prove that the answer was no.

Undecidability in Logic

Validity in propositional logic is decidable.

Validity in first-order predicate logic is undecidable, however, it is semi-decidable.

However, it is not the quantifiers *per se* that cause that problem. We cross the boundary to the undecidable only when the quantifiers appear together with predicates of arity greater than 1.

Monadic logic, that is, the fragment of predicate logic which does not allow predicates of arity 2 and above is decidable.

Rice's Theorem

We have this rather sweeping result:

Rice's Theorem: Every interesting semantic Turing machine property is undecidable!

A property P is interesting iff

$$P(M_1) \text{ and not } P(M_2)$$

for some Turing machines M_1 and M_2.

It is semantic iff

$$P(M_1) \text{ iff } P(M_2)$$

for all Turing machines M_1 and M_2 such that $L(M_1) = L(M_2)$.