COMPUTATION: DAY 5

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Review of the fourth day 1
2. The question of E_{TM}. 2
2.1. Machines that quit 2
2.2. How to search a double infinity 2
2.3. Finding the accepting string, when there is one. 3
3. Non-Recursive sets 3
3.1. It is undecidable if a Turing machine halts 4
3.2. It is undecidable if a language is non-empty 4
3.3. Rice's Theorem 5
3.4. It is undecidable if a language is Regular 5
4. Non-recursively enumerable sets 5
4.1. Emptiness of a language is co-RE 6
4.2. Equality of languages is neither RE nor co-RE 6
5. Merlin and Undecidability 7
5.1. Oracle Machines 7
5.2. Given an oracle for halting, we can enumerate non-equality 7
5.3. The question is $\mathcal{L}(M)$ regular 8

1. Review of the fourth day

Turing machines can be enumerated. While the enumeration function is dependent on many things: representations, some conventions; What we eventually have is the symbol $M_i(j)$, for the computation of the i-th Turing machine on the j-th input. This is universal as in all Turing machines are in the enumeration.

Given this we could show the set

$$A_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{N} \,|\, M_i(j) = T \,\}$$

is undecidable by construction a machine not among the enumerated machines, on the assumption that the set is decidable. However the set is Recursively Enumerable

because for those things in the set, the halting computation of $M_i(j)$ is sufficient to establish this fact.

If a set and its complement are Recursively Enumerable, the set is Recursive; as there are proceeders that recognize both elements in the set and not in the set. Hence the set $\neg A_{TM}$ is not Recursively Enumeratble.

2. The question of E_{TM}.

Monday, 27 March 2023

Consider the subset of the integers that is the set of indices for Turing machines for non-empty languages,

$$\neg E_{TM} = \{\, i \in \mathbb{N} \,|\, \mathcal{L}(M_i) \neq \emptyset \,\}$$

We shall this set to not be recursive, but to be recursively enumerable. In this section we will show it to be recursively enumerable.

To do so, we will first move some of the abstraction concerning non-halting into a quantification. Consider step-bounding our Turing machines. If after t steps the machine has not decided, it explicitly returns the non-halting symbol $\perp$. To accept or reject means that there exists a time bound t, and any time bound larger, in which the machine accepts or rejects. To not-halt is means that for all time bounds t the machine returns $\perp$.

2.1. **Machines that quit.** Let $M_i(j;t)$ denote the result of the i-th Turing machine computing the j-th input for up to t logical steps.
The value of $M_i(j;t)$ is defined in the natural way so as to agree with $M_i(j)$ in the following manner.

$$M_i(j) = \begin{cases} T & \exists t,\ M_i(j;t) = T \\ F & \exists t,\ M_i(j;t) = F \\ \perp & \forall t,\ M_i(j;t) = \perp \end{cases}$$

2.2. **How to search a double infinity.** Consider any surjective function

$$\begin{array}{rcl} \delta : \mathbb{N} & \to & \mathbb{N} \times \mathbb{N} \\ i & \mapsto & (i_1, i_2) \end{array}$$

such as the diagonal function d that proceeds with its first values $d(0), d(1), d(2), \ldots$ being,

$$(0,0), (1,0), (0,1), (2,0), (1,1), (0,2), (3,0), \ldots$$

2.3. **Finding the accepting string, when there is one.** Then

$$i \in \neg E_{TM} \iff \exists\, k \text{ such that } d(k) = (j, k) \text{ and } M_i(j; t) = T.$$

The formula on the right, with a single existential quantifier over the natural numbers, is a recognizer for co-emptiness. Essentially, we make attempts at accepting each input, never ignoring any input, but repeating each input with more steps, until some input on some number of steps accepts.

3. Non-Recursive sets

We show that non-emptiness is not recursive but showing that a Turing machine deciding non-emptiness implies a Turing machine that decides acceptance. This is done by creating an "easily computable" mapping that maps true instances of acceptance to machines with non-empty language, and other instance of acceptance to machines with empty languages.

Definition 3.1. A *recursive function* is and function $f : \Sigma^* \to \Sigma^*$ such that there exists an always-halting Turing machine M that computes the function in the following sense: When M is started with string s on its tape then it runs and halts with string $f(s)$ in its tape.

Definition 3.2. Given $A, B \subseteq \Sigma^*$, a *reduction of A to B* is a recursive function $f : \Sigma^* \to \Sigma^*$ that preserves the set inclusion,

(1) $\forall a \in \implies f(a) \in B$,
(2) $\forall a \notin A \implies f(a) \notin B$.

A reduction from A to B is denoted $A \leq_m B$.

Wednesday, 29 March 2023

Theorem 3.1. If $A \leq_m B$ and B is a recursive set, then A is a recursive set.

Proof: Let f be the reduction map. If B is recursive, then there as a recursive function $g : \Sigma^* \to \{T, F\}$. The function $g \circ f$ is recursive. Consider the Turing machine that computes this function and accepts if T is left on the tape and rejects of F is left on the tape. This Turing machine decides the set A.

Theorem 3.2. If $A \leq_m B$ and B is a recursively enumerable set, then A is a recursively enumerable set.

Proof: Similar to the above proof.

3.1. **It is undecidable if a Turing machine halts.** Define the halting problem for turing machines, H_{TM},

$$H_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{N} \mid M_i(j) \neq \perp \,\}$$

Consider the map $(i,j) \mapsto (f(i),j)$ which will act as,

$$M_{f(i)}(j) \equiv \text{if } M_i(j) \text{ then } T \text{ else } \perp$$

The function f is a recursive function which which retrieves the machine definition of machine i then operates on the definition to replace any transitions to a reject state to a transition to an infinite loop. It then gets the index $f(i)$ for this machine and outputs it on its tape, along with a copy of the original j.
It is a reduction,

$$\begin{aligned}
(i,j) \in A_{TM} &\implies M_i(j) = T \\
&\implies M_{f(i)}(j) = T \\
&\implies (f(i),j) \in H_{TM}. \\
(i,j) \notin A_{TM} &\implies M_i(j) \neq T \\
&\implies M_{f(i)}(j) = \perp \\
&\implies (f(i),j) \notin H_{TM}.
\end{aligned}$$

Note a little bit the game here. We are not attempting to distinguish rejecting from non-halting. Hence the gate to the non-halting behavior can it self not-halt, but only in the case where it would have invoked non-halting behavior anyway. When setting up a reduction, care is taken that this sort of submerging non-halting into the proper overall behavior for the machine.
This is the reduction

$$A_{TM} \leq H_{TM}$$

and since A_{TM} is not recursive then H_{TM} is not recursive.

3.2. **It is undecidable if a language is non-empty.** Consider the map $(i,j) \mapsto (f(i),j)$ which will act as,

$$M_{f(i,j)}(k) \equiv \text{if } M_i(j) \text{ then } T \text{ else } \perp$$

The function f is a recursive function which which retrieves the machine definition of machine i then operates on the definition to replace any transitions to a reject state to a transition to an infinite loop. It then adds states to write the given j onto the tape followed by a state transition to the start of the modified version of M_i. The function then index $f(i,j)$ for this machine and writes it to the tape.

$$\begin{array}{rcl}
(i,j) \in A_{TM} & \implies & M_i(j) = T \\
& \implies & \forall k\ M_{f(i,j)}(k) = T \\
& \implies & \mathcal{L}(M_{f(i,j)}) = \Sigma^* \\
& \implies & f(i,j) \in \neg E_{TM}. \\
(i,j) \notin A_{TM} & \implies & M_i(j) \neq T \\
& \implies & \forall k\ M_{f(i,j)}(k) = \bot \\
& \implies & \mathcal{L}(M_{f(i,j)}) = \emptyset \\
& \implies & f(i,j) \notin \neg E_{TM}
\end{array}$$

This is the reduction

$$A_{TM} \leq_m \neg E_{TM}$$

and since A_{TM} is not recursive then H_{TM} is not recursive.

Friday, 31 March 2023

3.3. **Rice's Theorem.** The reductions described are pretty general, so that any property of a language can be the right hand side of a reduction. Suppose $N(j)$ is the Turing machine with the property, and suppose the empty language does not have the property, then the construction

$$M_{f(i,j)}(k) \equiv \text{if } M_i(j) \text{ then } N(k) \text{ else } \bot$$

is a reduction $A_{TM} \leq_m P$. This gives Rice's theorem: any non-trivial property of a language is undecidable.

3.4. **It is undecidable if a language is Regular.** For instance, if $\mathcal{L}(N)$ is give by the regular expression $0^i\,1^i$, then this reduction shows that deciding where a language is not regular is undecidable. This reduction maps accepting instances $(i,j) \in A_{TM}$ to a non-regular context free language and non-accepting instances $(i,j) \notin A_{TM}$ to the the empty set, a regular language.

4. Non-recursively enumerable sets

The term *undecidable* refers to any non-recursive set. So far we have found two such sets, those that are *recursively enumerable* and those that are *co-recursively enumerable*, complements of recursively enumerable sets. We have employed reductions from a recursively enumerable set to a unknown set to show it is undecidable, and then given a recognizer for said unknown set to show it is recursively enumerable.

4.1. **Emptiness of a language is co-RE.** The set $\neg E_{TM}$ was shown to be undecidable. It was also shown to be recursively enumerable. There for its complement E_{TM} is co-recursively enumerable. It cannot be decided, recognized, or enumerated.

4.2. **Equality of languages is neither RE nor co-RE.** The question of whether two Turing machines differ require a more complicated statement that alternates quantifiers,

$$\neg EQ_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{M} \mid \exists (k, t_k)\ \forall t \geq t_k, M_i(k;t) \neq M_j(k;t) \,\}$$

The case here is when the difference between the two functions is a single point where M_i halts and M_j does not. On one level, we have definite evidence, that the disagreement was on input k, but also indefinite evidence, that for one machine a particular step count t suffices, but for the other machine we have to consider all step counts.

Note that a k specific lower bound for t is needed on the universal quantifier, $t > t_k$, since for small t both machines could still be computing and hence equal at that (k,t) point.

Consider the map $h_1(i,j) \mapsto (f(i,j), M_\emptyset)$ where,

$$M_{f(i,j)}(k) \equiv \text{if } M_i(j) \text{ then } T \text{ else } \bot$$

and $M_\emptyset$ is any Turing machine that accepts nothing. Previously it was described how f is a recursive function. It preserves truth from acceptance to non-equality.

$$\begin{aligned}
(i,j) \in A_{TM} &\implies M_i(j) = T \\
&\implies \mathcal{L}(M_{f(i,j)}) = \Sigma^* \\
&\implies M_{f(i,j)} \neq M_\emptyset \\
&\implies (f(i,j), M_\emptyset) \in \neg EQ_{TM}. \\
(i,j) \notin A_{TM} &\implies M_i(j) \neq T \\
&\implies \mathcal{L}(M_{f(i,j)} = \emptyset \\
&\implies M_{f(i,j)} = M_\emptyset \\
&\implies (f(i,j), M_\emptyset) \notin \neg EQ_{TM}
\end{aligned}$$

Consider the very similar map $h_2(i,j) \mapsto (f(i,j), M_{\Sigma^*})$ where M_{Σ^*} is the machine that accepts everything. It preserves truth from acceptance to equality,

$$\begin{aligned}
(i,j) \in A_{TM} &\implies M_i(j) = T \\
&\implies (f(i,j), M_{\Sigma^*}) \in EQ_{TM}. \\
(i,j) \notin A_{TM} &\implies M_i(j) \neq T \\
&\implies (f(i,j), M_{\Sigma^*}) \notin EQ_{TM}
\end{aligned}$$

So we have two reductions,

$$A_{TM} \leq_m \neg EQ_{TM}, \;\; A_{TM} \leq_m EQ_{TM}$$

so neither can be recursive. As reductions are stable by complementing both sides, we also have,

$$\neg A_{TM} \leq_m EQ_{TM}, \;\; \neg A_{TM} \leq_m \neg EQ_{TM}$$

So neither can be recursively enumerable either.

These are sets that cannot be decided, and further neither the set nor its complement can be recognized.

5. Merlin and Undecidability

In the theory of languages, we previously introduced Merlin, the oracle that would honest and accurately answer questions. In the case of a Turing machine, if it were non-deterministic Merlin would provide guidance towards an accepting computation. One thing Merlin will not do, is provide an infinite stream of advice.

Therefore, if Merlin wants to be efficient, he either begins to give advice, in which case we immediately know the Turing machine will accept, or immediately refuses to give any advice, in which case we know the Turing machine will reject or not halt.

Therefore Merlin is truly magical because he solves the halting problem.

5.1. **Oracle Machines.** An oracle machine is a Turing machine with the ability to ask for Merlin's advice. It can be in the form of a an additional oracle tape, but more often it is modeled with a message tape in which the Turing machine writes a question, enters a query Merlin state, during which Merlin replaces the contents of the message tape with the response, and the Turing machine exist the query state to some next state.

Merlin is an oracle for H_{TM}.

A Turing machine with an oracle for H_{TM} is recursive for the halting set, but also for the acceptance set and the non-emptiness set. All of these problems become decidable if we can ask Merlin whether or not machine i halts on input j.

5.2. **Given an oracle for halting, we can enumerate non-equality.** The problem of non-equality becomes recursively enumerable when the Turing machine can query an oracle for H_{TM}.

Suppose the two machines are M_i and M_j. For each k,

(1) Query the oracle to determine if either or both machines halt.
(2) If they both do not halt, continue on.
(3) If only one halts, accept. They machines are not equal.

(4) If they both halt, compute to determine if they halt with the same outcome; continue on if they do, and accept if they do not.

If the machines are unequal, eventually the k at which they are unequal will be tested. If they are equal, the algorithm will not halt. Hence we have a recognizer for non-equality.

5.3. **The question is $\mathcal{L}(M)$ regular.** However, even Merlin cannot recognize regular languages. That problem, which can be shown undecidable by the reduction $A_{TM} \leq_m REG$ given by the recursive function f acting as,

$$M_{f(i,j)}(k) \equiv N(k) \text{ or } \big(\text{ if } M_i(j) \text{ then } T \text{ else } \bot \big)$$

where $\mathcal{L}(N)$ given by the regular express $0^i 1^i$ and the "or" is short-circuited as in C language. Previously in these notes the complement was shown undecidable as an application of Rice's theorem.

Since A_{TM} reduces to both REG and $\neg REG$, REG is neither RE or co-RE, such as EQ_{TM}. However, even relative to A_{TM} these languages are not RE. The full writing of requires additional alternation of quantifiers than was required when writing the set of language equality,

$$REG = \{\, i \mid \exists j \; \forall k \; \exists t \; F_j(k) = M_i(k;t) \,\}$$

Where $\mathbb{N} \to \{\, F_i \,\}$ is an enumeration of all finite state automata's, and since such machines always halt, the quantification over steps is existential (it's only a matter of when, not if, when equality is assumed).