CS 455/555: Computability theory

Stefan D. Bruda

Fall 2020

The Church thesis

- It has been shown that all the formalisms that model general computations (primitive recursive functions, the lambda calculus, unrestricted grammars, the random access machine, etc.) are equivalent with each other
- They must thus be equivalent to the general notion of computation—the Church thesis, proposed by... Stephen Kleene (a student of Alonzo Church) in 1943
- We can thus analyze algorithms, computations, and problems exclussively in terms of Turing machines
- An algorithm is a Turing machine that decides a language (problem)
- A program would then be a Turing machine that semidecide a language/problem
- How about a computer? This is going to be also a Turing machine
  - Such a machine will take as input another Turing machine and will execute it on an input also provided
  - We call it the Universal Turing machine
  - We need to uniformly encode Turing machines and their input strings

- Choose a uniform encoding for states, such as $q\langle n\rangle$, where $\langle n\rangle$ is a binary representation of fixed length
  - Make it long enough so that we have room for all the states
  - Also specify specific encodings for the initial state and the halt state, e.g. $\text{enc}(s) = q00\ldots0$, $\text{enc}(h) = q11\ldots1$
- Choose an encoding for tape symbols such as $a\langle n\rangle$, with $\langle n\rangle$ as above (and long enough to include all the tape symbols and also L and R)
  - Identify the special symbols $\#$, $\triangleright$, L, and R as being, say the first four symbols in the encodings
  - For example this is an acceptable encoding of inputs over $\{a, b\}$:

$$\begin{array}{lcl lcl lcl} \text{enc}(\#) & = & a000 & \text{enc}(\triangleright) & = & a001 & \text{enc}(L) & = & a010 \\ \text{enc}(R) & = & a011 & \text{enc}(a) & = & a100 & \text{enc}(b) & = & a101 \end{array}$$

- A transition can easily be encoded; for example: $\text{enc}((q, a, h, L)) = q010a100q111a010$
  - A whole transition relation is then encoded as the concatenation of all the transitions therein
- Given the conventions above we have

$$\text{enc}(M) = \text{enc}(\Delta) \text{ for any } M = (K, \Sigma, \Delta, s, \{h\})$$

- The universal Turing machine is a machine U such that $U(\text{enc}(M)\#\text{enc}(w)) = \text{enc}(M(w))$ for any Turing machine M and input w for M
- Computation easily accomplished with three tapes:
  - First tape is the working tape: U will move $\text{enc}(M)$ onto the second tape and the first tape then contains $\text{enc}(w)$ as manipulated by M
  - The head of the first tape keeps scanning the prefix a of the symbol currently scanned by the head of M
  - The second tape will contain $\text{enc}(M)$ copied from the first tape at the beginning and does not change
  - The third tape is initialized with $q00\ldots0$ (the encoding of the initial state) and will keep storing the current state
  - A step of M is simulated by U as follows:
    1. U finds the current symbol (first tape) and the current state (third tape)
    2. U guesses nondeterministically the transition (second tape) to be applied
    3. The transition is applied (the first and third tapes are changed accordingly)
    4. If the third tape is $q11\ldots1$ (the halt state) then U halts, otherwise it repeats from Step 1

Recursive versus recursivley enumerable languages

- Are recursive languages the same as recursively enumerable languages?
- If so, all problems that can be formulated computationally admit algorithms (are solvable computationally)
- Unfortunately this turns out not to be the case
- Simple diagonalization argument. Crux:
  - Let halt(P, x) = halts iff P halts on input x
  - Let diagonal(x) = if halt(x, x) then diagonal(x) else halt
  - Does diagonal halt?

The halting problem

- The halting problem is represented by the language

$$H = \{\text{enc}(M)\#\text{enc}(w) : M \text{ halts on } w\}$$

- H is recursively enumerable, for indeed it is semidecided by U
- Suppose H is recursive and decided by M_H
  - If so, then all the recursively enumerable languages are recursive!
  - Indeed, consider a language L semidecided by M; for each string w we produce $\text{enc}(M)\#\text{enc}(w)$ and we launch M_H, thus deciding whether $w \in L$
  - H is complete for recursively enumerable languages
- Let now $H_1 = \{\text{enc}(M) : M \text{ halts on } \text{enc}(M)\}$
  - H is recursive then H_1 is also recursive
  - Indeed, for any $\text{enc}(M)$ received as input we duplicate it (thus obtaining $\text{enc}(M)\#\text{enc}(M)$) and then we launch M_H
- Since H_1 is recursive then so is $\overline{H_1}$ (recursive languages are closed under $\overline{\cdot}$)

The halting problem (cont'd)

- $\overline{H_1} = \{w$: either w is not the encoding of a Turing machine, or $w = \text{enc}(M)$ such that M does not halt on input $w\}$
- Since $\overline{H_1}$ is recursive then it is also recursively enumerable
- Let M^* be the Turing machine that semidecides $\overline{H_1}$
- Is it the case that $\text{enc}(M^*) \in \overline{H_1}$?
  - From the definition of $\overline{H_1}$: $\text{enc}(M^*) \in \overline{H_1}$ iff M^* does not halt on $\text{enc}(M^*)$
  - From the definition of M^*: $\text{enc}(M^*) \in \overline{H_1}$ iff M^* accepts (halts on) $\text{enc}(M^*)$
  - Contradiction!

Theorem

Recursive languages are a strict subset of recursively enumerable languages

Theorem

Recursively enumerable languages are not closed under complementation

- H_1 is recursively enumerable (decided by U) but $\overline{H_1}$ is not

- There are more recursively enumerable languages/problems that are not recursive
- These are easily found via reductions
- Let $L_1, L_2 \in \Sigma^*$; a reduction from L_1 to L_2 is the recursive function $\tau : \Sigma^* \to \Sigma^*$ such that $w \in L_1$ iff $\tau(w) \in L_2$

Theorem

If L_1 is not recursive and there exists a reduction from L_1 to L_2 then L_2 is not recursive

  - Suppose L_2 is recursive so that M_2 decides L_2
  - Let M_τ be the Turing machine that computes τ, the reduction from L_1 to L_2
  - Then the machine $M_\tau M_2$ decides L_1, a contradiction

- To prove that a certain language L is not recursive all we need is to provide a reduction from a known non-recursive language to L

- A whole bunch of them, check out Sections 5.4, 5.5, and 5.6 (the latter very important)
- Most interesting problems about Turing machines turn out to be undecidable

Theorem (Rice's theorem)

Let P be a property over Turing machines. If P is

- *non-trivial (there exists at least one Turing machine that has P and at least one Turing machine that does not have it) and*
- *extensional (if a Turing machine that decides L has P then all the Turing machines that decide L have P)*

then P is undecidable

- Proof on p. 270

SOME UNDECIDABLE PROBLEMS ABOUT TURING MACHINES

1. Does M halt on w?
2. Does M halt on an empty tape?
   - Reduction from $H = \{\text{enc}(M)\#\text{enc}(w) : M \text{ halts on } w\}$ to $L = \{\text{enc}(M) : M \text{ halts on } \varepsilon\}$
   - Given M, $w = w_1 w_2 \cdots w_n$, the reduction produces M_w which starts with an empty tape, writes w and launches M, i.e., $M_w = w_1 R w_2 R \cdots w_n R M$
3. Is there any input string on which M halts?
   - Similar reduction from H
   - Given M, w, the reduction produces M_w that erases w from the input tape, guesses nondeterministically a string x and launches M (on x)
4. Given a Turing machine M that semidecides a language L, is L regular? context-free? recursive?
5. Given a Turing machine M that semidecides a language L, is L empty?
6. Given two Turing machines, do they decide the same language?

- Algorithm = decides a recursive language
- Solvable (decidable) problem = recursive language
- Problem in general = recursively enumerable language
- A recursively enumerable language L is recursive iff both L and $\overline{L}$ are recursively enumerable
- Recursive languages are closed under complementation
- Recursively enumerable languages are not closed under complementation