Computability Theory Cheat Sheet

Alessandro Cheli - 2020 - University of Pisa

Algorithm Definition

- Finite set of available instructions
- Made of finite instructions
- Discrete steps taking finite time
- Each step depends only on the previous ones
- No limit to steps or space during execution

Turing Machine

A Turing Machine M is (Q, Σ, δ, q_0)

$Q(\ni q_i), h \notin Q$	Set of states, h halting state
$\Sigma(\ni \sigma, \sigma', \ldots)$	Set of symbols
$q_0 \in Q$	Initial state
$\delta \subseteq (Q \times \Sigma) \times (Q \cup \{h\} \times \Sigma \times \{L, R, -\})$	is the transition function
if $(q, \triangleright, q', \sigma, D) \in \delta$	then $\sigma = \triangleright, D = R$

Σ^* is the free monoid generated from Σ. " $\cdot$ " is the associative string concatenation operator. ϵ is the empty string and identity element.

Configuration

A configuration C of a TM M is a quadruple:
$(q, u, \sigma, v) \in (Q \cup \{h\}) \times \Sigma^* \times \Sigma \times \Sigma^F)$ where $\Sigma^F = \Sigma^* \cdot (\Sigma \backslash \{\#\}) \cup \{\epsilon\}$.
σ is the currently selected symbol under the head of the machine. Short notation is $(q, u\underline{\sigma}v)$ or (q, w) when position is not needed.

Computations

A computation is a finite succession of computation steps

$(q, w) \to (q', w')$	Computation step
$(q_0, w) \to^* (q', w')$	Reflexive and transitive closure of comp. steps.
$(q_0, w) \to^n_M (q', w')$	Machine M computes w' in n steps.
$\downarrow, \uparrow$	Converges and diverges

T(uring)-Computability

Let $\Sigma, \Sigma_0, \Sigma_1$ be alphabets, let $\#, \triangleright \notin \Sigma_0 \cup \Sigma_1$ and $\Sigma_0 \cup \Sigma_1 \subset \Sigma$.
Let f be a function $f : \Sigma_0^* \to \Sigma_1^*$.

$$f \text{ is turing computable by a machine } M \Leftrightarrow$$
$$\forall w \in \Sigma_0^* \ : f(w) = z \Leftrightarrow (q_0, \underline{\triangleright} w) \to^*_M (h, \triangleright z \underline{\#})$$

While Computability

A command C computes $f : \text{Var} \to \mathbb{N} \Leftrightarrow$

$$\forall \sigma \in \text{Var} \to \mathbb{N} \ : \ f(x) = n \Leftrightarrow \langle C, \sigma \rangle \to^* \sigma' \wedge \sigma'(x) = n$$

Problems and Functions

Goldbach Conjecture

$\forall m > 1 \ : \ 2m = p_1 + p_2$ with p_1, p_2 prime numbers.

Total Function

$f : A \to B$, subset of $A \times B$ is total $\Leftrightarrow$

$\forall a \in A \ \exists b \in B \ : (a, b) \in f$	f is defined everywhere
$(a, b), (a, c) \in f \Rightarrow b = c$	uniqueness

Partial Function

$f : A \to B$, subset of $A \times B$ is a *partial* function $\Leftrightarrow$

$(a, b), (a, c) \in f \Rightarrow b = c$	uniqueness
$\exists$ at least a $b \in B \ : \ f(a) = b$	not required to be defined everywhere

Primitive Recursive Functions

Are the minimum class of functions $\mathcal{C}$, from $\mathbb{N}^n$ with $n \geq 0$ to $\mathbb{N}$.

1: Zero	$\lambda x_1, \ldots, x_k.0$ with $k \geq 0$
2: Successor	$\lambda x.x + 1$
3: Identity (projections)	$\lambda x_1, \ldots, x_k.x_i$ with $1 \leq i leq k$
4: Composition	$\lambda x_1, \ldots, x_m.h(g_1(\ldots, x_m), \ldots, g_k(\ldots, x_m))$ h func. in k variables, $g_1, \ldots, g_k$ funcs. in m variables. The composition is p.r.

Recursion Axiom

Let h be a p.r. func. in $k+1$ variables, g be a p.r. func in $k-1$ variables. The function f in k variables defined as follow is then p.r.

$$f = \begin{cases} f(0, x_2, \ldots, x_k) &= g(x_2, \ldots, x_k) & \text{final step} \\ f(x_1 + 1, \ldots, x_k) &= h(x_1, f(x_1, \ldots, x_k), x_2, \ldots, x_k) & \text{iteration} \end{cases}$$

This represents a **FOR** loop. x_1, the first variable of f is the decrementing counter. g is the final step. h represents loop steps. This is guaranteed to terminate. One can define basic maths primitives with p.r. functions.

μ-Recursive (or General Recursive) Functions

The class of μ-recursive functions extends the class of p.r. functions with an additional condition to also include partial functions. We use letters φ and ψ to denote most μ.r. functions.

6: Minimization — if $\varphi(x_1, \ldots, x_n, y)$ is μ.r. in $n+1$ variables then ψ in n variables is μ.r. if it is defined by:

$$\psi(x_1, \ldots, x_n) = \mu y[\varphi(x_1, \ldots, x_n, y) = 0) \\ \forall z \leq y.\varphi(x_1, \ldots, x_n, z) \downarrow]$$

The μ operator corresponds to `while` loops. It starts with $y = 0$ and increments it until the φ equals to 0 (loop guard), and returns y. If an upper bound is placed on y then ψ may be p.r. If φ is p.r. the additional convergence condition is guaranteed. Diagonalization cannot be applied directly to μ.r. funcs. because $\varphi_n(n)$ may diverge. Not all μ.r. functions can be extended to total functions. A function is μ-computable if a μ.r. definition can be given.

Gödel Numbering

The characteristic function of the set of prime numbers is p.r. The exponents of the unique prime factorization of any natural numbers are given by p.r. functions. It follows that every sequence of natural numbers can be encoded as a natural number, by using the numbers as the exponents of the prime factoring. **This encoding is injective but not surjective**: an effective encoding must be surjective. Gödel gave a better, surjective encoding. It follows that such *encoding and decoding functions* are p.r. All of Turing Machines, FOR/WHILE programs, p.r. and μ-recursive functions can be effectively encoded and decoded as natural numbers by treating the syntactic symbols as distinct natural numbers. Computations (successions of configurations) can also be encoded and decoded

Ackermann Function

The Ackerman function A is effectively computable but grows faster than any possible p.r. function. It calls itself a number of times that is not possible to achieve with a p.r. definition.

$$A(0, 0, y) = y \qquad A(0, x + 1, y) = A(0, x, y) + 1$$
$$A(1, 0, y) = 0 \qquad A(z + 2, 0, y) = 1$$
$$A(z + 1, x + 1, y) = \qquad A(z, A(z + 1, x, y), y)$$

Church-Turing Thesis

> The (intuitively/effectively) computable functions are all and only the (partial), T-Computable functions

T-computable, μ-computable and WHILE-computable functions are the same class of functions. It is the same for all the other turing complete formalisms: what is computable does not change.

Classical Results

Note. Given an effective numbering, φ_i denotes the partial function computed by TM M_i. For $i \neq j$, functions may be semantically equal: $\varphi_i = \varphi_j$, but surely $M_i \neq M_j$.

Theorem. *The class of Primitive Recursive functions (and extensions) does not contain all the effectively computable functions:*

Proof. reductio ad absurdum

1: Encode and enumerate p.r. functions	$f_0, \ldots, f_n$. There are $\mathbb{N}$
2: Define diagonal func. $g(x) = f_x(x) + 1$	Intuitively computable
3: $\forall n.g(n) \neq f_n(n) \Rightarrow \forall n.g \neq f_n$	g doesn't appear in list

Implies there is no formalism for expressing only and all total functions. □

Theorem. *Computable functions are $\#(\mathbb{N})$. Total computable functions are $\#(\mathbb{N})$. There exists functions that are not computable.*

Proof. Computable functions can be given a Gödel numbering. Therefore there exists a 1-1 mapping with natural numbers. All the possible total functions $\{f : \mathbb{N} \to \mathbb{N}\}$ are $2^{\#(\mathbb{N})}$. □

Theorem. ***Padding Lemma**: every computable function φ_i has $\#(\mathbb{N})$ indexes. $\forall i. \exists A_i$ infinite set of indexes such that $\forall y \in A_i.\varphi_y = \varphi_i$*

Proof. If a program P_j in WHILE computes φ_i, one can build $\#(\mathbb{N})$ other programs that compute the same thing, semantically equal to φ_i by just adding a `skip` command at the end of the program. The new program's index will change because an effective numbering depends only on the syntactic description of the program. □

Theorem. ***Kleene I: Normal form theorem**: There exist a predicate $T(i, x, y)$ and a function $U(y)$ that are total and computable, such that $\forall i, x.\varphi(x) = U(\mu y.T(i, x, y))$*

Proof. Define T as true $\Leftrightarrow$ y is encodes a successful computation of the TM M_i on input x. Check if each step encoded in y is a step of $M_i(x)$ and if it terminates. This procedure is total and p.r.. Define U such that it retrieves the word on tape in the last configuration of y (the result) and return its encoding. This is a p.r. procedure □

Corollary. *A function f is T-computable $\Leftrightarrow$ it is μ-computable*

Theorem. ***Enumeration**: $\exists z. \forall i, x. \ \varphi_z(i, x) = \varphi_i(x)$*

Proof. Guarantees that a formalism that expresses all computable function has an universal algorithm. Therefore there exists an *Universal Turing Machine*. $\varphi_z(i, x) = U(\mu y.T(i, x, y)) = \varphi_i(x)$ □

Theorem. ***s-1-1**: There exists a total, injective computable function s^1_1 of 2 variables such that $\forall x, y. \ \varphi_{s^1_1(i,x)}(y) = \lambda y.\varphi_i(x, y)$.*

Proof. To compute $\varphi_{s^1_1(i,x)}(y)$, retrieve M_i, set the initial state $M_i(x, y)$ (algorithmic procedure guaranteed to terminate). By the *Church-Turing Thesis* there exists $s = s^1_1$. If s was not injective, build s' such that $\varphi_{s(i,x)} = \varphi_{s'(i,x)}$ such that s' is strictly increasing, which means injective. This can be done because from the padding lemma we have that there are $\#(\mathbb{N})$ indexes of TMs that compute $s(i, x)$. □

The Parameter Theorem entails partial evaluation as we know it.

Theorem. ***s-m-n Parameter Theorem**: $\forall m, n \geq 0. \ \exists s^m_n$ injective total computable function in $m + 1$ variables . $\forall i, x_1, \ldots, x_m$.*

$$\varphi^{(n)}_{s^m_n(i,x_1,\ldots,x_m)} = \lambda y_1, \ldots, y_n. \ \varphi^{(m+n)}_i(x_1, \ldots, x_m, y_1, \ldots, y_n)$$

Theorem. ***Expressiveness**: A formalism is **Turing-equivalent**, or universal (computes all and only T-computable functions) $\Leftrightarrow$ it has an universal algorithm (enumeration theorem is valid) and the parameter theorem is valid.*

Theorem. ***Kleene II: Recursion**: $\forall f$ total computable. $\exists n$ such that $\varphi_n = \varphi_{f(n)}$*

Proof. Let there be a computable function $\psi_i(u, x)$ defined as

$$\psi_i(u, x) = \varphi_{d(u)}(x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \text{if } \phi_u(u) \downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that $d(u) = \lambda u.s(i, u)$. Since $d(u)$ is total and computable (by the *parameter theorem*), and f is total and computable by hypothesis, $f(d(u))$ is total and computable as well. Therefore, it has an index, let it be v: $\varphi_v(x) = f(d(x))$. It follows that $\varphi_v(v) \downarrow$. Let now $n = d(v)$. We have that

$$\varphi_n = \varphi_{d(v)} = \varphi_{\varphi_v(v)} = \varphi_{f(d(v))} = \varphi_{f(n)}$$

n is called *fixed point.* □

Theorem. *Considering the hypothesis of the Recursion Theorem, there are $\#(\mathbb{N})$ fixed points of f, and the fixed point is computable from a total injective function from the index of f.*

Proof. (Injectivity of g is not considered). Let $h(x)$ be a total computable function such that $\forall n. \ \varphi_{h(x)}(n) = \varphi_x(d(n))$. Then $g(x) = d(h(x))$ □

Reductions

A (decision) problem (a set) A reduces to B with *reduction* f, in symbols $A \leqslant_f B$ when $a \in A \Leftrightarrow f(a) \in B$. We also have that $A \leqslant_f B \Leftrightarrow \overline{A} \leqslant_f \overline{B}$ because $x \in \overline{A} \Leftrightarrow x \notin A \Leftrightarrow f(x) \notin B \Leftrightarrow f(x) \in \overline{B}$.

Reduction Relations

A relation of reductions ($\leqslant_F$) can be defined on a class of functions F as follows, note that $\leqslant_F$ is also a partial pre-order.

$$A \leqslant_F B \Leftrightarrow \exists f \in F. \ A \leqslant_f B$$

Properties

Let $\mathcal{D}$ and $\mathcal{E}$ two classes of problems such that $\mathcal{D} \subseteq \mathcal{E}$, and $\mathcal{D} \subseteq \mathcal{H}$. A reduction relation $\leqslant_F$ classifies $\mathcal{D}$ and $\mathcal{E} \Leftrightarrow \forall A, B, C$ (problems):

1: Reflexivity	$A \leqslant_F A$
2: Transitivity	$A \leqslant_F B, B \leqslant_F C \Rightarrow A \leqslant_F C$
3: $\mathcal{D}$ closed by reduction	$A \leqslant_F B, B \in \mathcal{D} \Rightarrow A \in \mathcal{D}$
4: $\mathcal{E}$ closed by reduction	$A \leqslant_F B, B \in \mathcal{E} \Rightarrow A \in \mathcal{E}$

Equivalently:

1: Identity	$\text{id} \in F$
2: Closed by composition	$f, g \in F \Rightarrow f \circ g \in F$
3:	$f \in F, B \in \mathcal{D} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{D}$
4:	$f \in F, B \in \mathcal{E} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{E}$

Hard and Complete Problems

If $\leqslant_F$ classifies $\mathcal{D}$ and $\mathcal{E}$, then $\forall$ problems A, B, H.

- $A \leqslant_F B \wedge B \leqslant_F A \Rightarrow A \equiv B$
- H is $\leqslant_F$-hard for $\mathcal{E}$ if $\forall A \in \mathcal{E}. \ A \leqslant_F H$
- H is $\leqslant_F$-complete for $\mathcal{E}$ if $\forall A \in \mathcal{E}. \ A \leqslant_F H \wedge H \in \mathcal{E}$

Theorem. *If $\leqslant_F$ classifies $\mathcal{D}$ and $\mathcal{E}$, $\mathcal{D} \subseteq \mathcal{E}$ and C is complete for $\mathcal{E}$ then $C \in \mathcal{D} \Leftrightarrow \mathcal{D} = \mathcal{E}$*

Proof. ($\Leftarrow$) is obvious. ($\Rightarrow$): Let $C \in D$ and $A \in \mathcal{E}$. By completeness, $A \leqslant_F C$ and $A \in \mathcal{D}$, then $\mathcal{E} \subseteq \mathcal{D}$ which implies $\mathcal{E} = \mathcal{D}$ □

Theorem. ***Completeness is "transitive" for reductions**: If A is complete for $\mathcal{E}$, $A \leqslant_F B$ and $B \in \mathcal{E}$ then B is complete for $\mathcal{E}$.*

Undecidable Problems

A set I is **recursive** if its characteristic function χ_I is total computable. $\chi_I(x) = 1 \Leftrightarrow x \in I$ and 0 otherwise. A set I is **recursively** (or computably) **enumerable** $\Leftrightarrow \exists i. \ I = \{x \mid \varphi_i(x) \downarrow\}$. That function is said semi-characteristic and the problem is said semi-decidable. We define the classes R for recursive sets and RE for recursively enumerable sets.

$$I \in R \Rightarrow I \in RE \qquad A \in R \Leftrightarrow \overline{A} \in R \qquad I, \overline{I} \in RE \Leftrightarrow I, \overline{I} \in R$$

Theorem. *I is RE $\Leftrightarrow I = \emptyset$ or I is the range of a total computable function.*

$$\boxed{K = \{x \mid \varphi_x(x) \downarrow\}. \ K \text{ is RE but not R}}$$

Proof. by *reductio ad absurdum*: Suppose by absurd that χ_K was total and computable, for K to be recursive. Then
$f = \left(\left\{\varphi_x(x) + 1 \quad \text{if } x \in K;\ 0 \text{ otherwise}\right)\right.$ would be total and computable. We obtain a contradiction because $\forall x. \ f(x) \neq \varphi_x(x)$. □

We define rec as the class of total computable functions. Note that $\overline{K} \not\leqslant_{\text{rec}} K$ and $K \not\leqslant_{\text{rec}} \overline{K}$

$$\boxed{\begin{array}{l}\text{Halting Problem:} \\ K_0 = \{(x, y) \mid \varphi_y(x) \downarrow\}. \ K_0 \text{ is not R}\end{array}}$$

Proof. $x \in K \Leftrightarrow (x, x) \in K_0$. If K_0 was R, K would too. □

Theorem. *K **is $\leqslant_{rec}$-complete for RE***

Proof. Let $A \in RE$. $x \in A \Leftrightarrow f(x) \in K$ with $f(x) = \lambda x.s(i, x)$

$$\begin{aligned} A &= \{x \mid \varphi_j(x) \downarrow\} = \{x \mid \varphi_i(x, y) \downarrow\} \\ &= \{x \mid \varphi_{s(i,x)}(y) \downarrow\} = \{x \mid \varphi_{s(i,x)}(s(i, x)) \downarrow\} \\ &= \{x \mid s(i, x) \in K\} \end{aligned}$$

□

A is a set of indexes representing functions
$\Leftrightarrow \forall x, y. \ x \in A \wedge \varphi_x = \varphi_y \Rightarrow y \in A$.

Theorem. *If A is a set of indexes representing functions, with $\emptyset \neq A \neq \mathbb{N}$, then $K \leqslant_{rec} A$, or $K \leqslant_{rec} \overline{A}$.*

Proof. Let i_0 be the index of the *always undefined* function $\varphi_{i_0}(x)$, such that $i_0 \in \overline{A}$ (if not proceed symmetrically). **Let us prove that $K \leqslant_{\textbf{rec}} A$**: Pick $i_1 \in A$. We have that $\varphi_{i_0} \neq \varphi_{i_1}$ by prev. definition.

$$\psi(x, y) = \varphi_{f(x)}(y) = \begin{cases} \varphi_{i_1}(y) & \text{if } x \in K \\ \text{undefined} = \varphi_{i_0}(y) & \text{otherwise} \end{cases}$$

With $f(x) = \lambda x.s(i, x)$, we now have that
$x \in K \Rightarrow \varphi_{f(i)} = \varphi_{i_1} \Rightarrow f(x) \in A$ and we also have that
$x \notin K \Rightarrow \varphi_{f(x)} = \varphi_{i_0} \Rightarrow f(x) \in \overline{A} \Rightarrow f(x) \notin A$ □

Corollary. *If $K \leqslant_{rec} A \wedge K \leqslant_{rec} \overline{A}$ then A **is not RE**.*

Theorem. ***Rice**: Let $\mathcal{A}$ is a class of computable functions. The set $A = \{n \mid \varphi_n \in \mathcal{A}\}$ is recursive $\Leftrightarrow \mathcal{A} = \emptyset$ or $\mathcal{A}$ is the class of all computable functions.*

Proof. If $\emptyset \neq \mathcal{A} \neq \mathbb{N}$ then apply the previous theorem. If A was recursive then also K would be recursive. *Reductio ad absurdum.* In the other two cases proof is obvious. □

Other undecidable (not RE) problems: semantic equality of two functions, $FIN = \{x \mid \text{dom}(\varphi_x) \text{ is finite}\}$, $INF = \overline{FIN} = \{x \mid \text{dom}(\varphi_x) \text{ is infinite}\}$, $TOT = \{x \mid \varphi_x \text{ total}\}$, $REC = \{x \mid \text{dom}(\varphi_x) \text{ is recursive}\}$, $CONST = \{x \mid \varphi_x \text{ is total and constant}\}$, $EXT = \{x \mid \varphi_x \text{ can be extended to total computable}\}$,