Computability
Decidability
Incompleteness

Second Lecture

11 April 2023

Part C : Computability

§ C.1 Register Machines

4.1 Register machines

Let Σ be an alphabet and Q a non-empty finite set whose elements we shall call *states*. A tuple of the form

$$
\begin{gathered}
(0, k, a, q) \in \mathbb{N} \times \mathbb{N} \times \Sigma \times Q, \\
(1, k, a, q, q') \in \mathbb{N} \times \mathbb{N} \times \Sigma \times Q \times Q, \\
(2, k, q, q') \in \mathbb{N} \times \mathbb{N} \times Q \times Q \text{ or} \\
(3, k, q, q') \in \mathbb{N} \times \mathbb{N} \times Q \times Q
\end{gathered}
$$

is called a (Σ, Q)-*instruction*. For improved readability, we write

$$
\begin{aligned}
+(k, a, q) &:= (0, k, a, q), && (\text{"add"}) \\
?(k, a, q, q') &:= (1, k, a, q, q'), && (\text{"check"}) \\
?(k, \varepsilon, q, q') &:= (2, k, q, q') \text{ and} && (\text{"check"}) \\
-(k, q, q') &:= (3, k, q, q') && (\text{"remove"})
\end{aligned}
$$

Instruction	Interpretation
$+(k, a, q)$	"Add the letter a to the content of register k and go to state q."
$?(k, a, q, q')$	"Check whether the last letter in register k is a; if so, go to state q; otherwise, go to state q'."
$?(k, \varepsilon, q, q')$	"Check whether register k is empty; if so, go to state q; otherwise, go to state q'."
$-(k, q, q')$	"Check whether register k is empty; if so, go to state q; otherwise, remove the final letter of its content and go to state q'".

Definition A Σ-register machine is a triple (Σ, Q, P) where Σ is an alphabet, Q is a nonempty finite set of states, P is a function with domain Q s.t. for all $q \in Q$, $P(q)$ is an instruction.

Also assume that $q_H \neq q_S \in Q$.

HALT STATE START STATE

From Lecture #1

Upper register index:

largest k occurring in a RM.

If M is a RM with v.r.i. n, we interpret it as a machine with

$n+1$ storage units

REGISTERS

each of which can contain a word $w \in \Sigma^*$.

Therefore we say that elements of

$$Q \times \mathbb{W}^{n+1}$$

$$(q, w_0, \dots, w_n)$$

are configurations or snapshots of such a machine.

STATE of the configuration

REGISTER CONTENT of configuration

Goal: Define the transformation of a configuration by a RM $M = (\Sigma, Q, P)$.

If C is a configuration and $M = (\Sigma, Q, P)$ is a RM, we say that M transforms C to C' if

Case 1. If $P(q) = +(k, a, q')$ and $C' = (q', w_0, ..., w_{k-1}, w_k a, w_{k+1}, ..., w_m)$.

Case 2. If $P(q) = ?(k, a, q', q'')$,

Subcase 2a. $w_k = wa$ for some w and $C' = (q', w_0, ..., w_m)$ or

Subcase 2b. $w_k \neq wa$ for any w and $C' = (q'', w_0, ..., w_m)$.

Case 3. If $P(q) = ?(k, \varepsilon, q', q'')$,

Subcase 3a. $w_k = \varepsilon$ and $C' = (q', w_0, ..., w_m)$ or

Subcase 3b. $w_k \neq \varepsilon$ and $C' = (q'', w_0, ..., w_m)$.

Case 4. If $P(q) = -(k, q', q'')$,

Subcase 4a. $w_k = \varepsilon$ and $C' = (q', w_0, ..., w_m)$ or

Subcase 4b. $w_k = wa$ for some a and $C' = (q'', w_0, ..., w_{k-1}, w, w_{k+1}, ..., w_m)$.

$C = (q, w_0, \dots, w_m)$.

Observe

Transforming C to C' is a function on the set of configurations.

Instruction	Interpretation
$+(k, a, q)$	"Add the letter a to the content of register k and go to state q."
$?(k, a, q, q')$	"Check whether the last letter in register k is a; if so, go to state q; otherwise, go to state q'."
$?(k, \varepsilon, q, q')$	"Check whether register k is empty; if so, go to state q; otherwise, go to state q'."
$-(k, q, q')$	"Check whether register k is empty; if so, go to state q; otherwise, remove the final letter of its content and go to state q'".

Suppose M is a RM with u.r.i. m and $\vec{w} \in \mathbb{W}^{m+1}$.

Remember that we had q_S, q_H called start state and halt state in Q.

We call

$(q_S, \vec{w})$ the START CONFIGURATION w/ INPUT $\vec{w}$

and define

$$C(0, M, \vec{w}) := (q_S, \vec{w})$$

$$C(k+1, M, \vec{w}) := C'$$

if M transforms $C(k, M, \vec{w})$ to C'.

This infinite seq. of configurations is called the computation sequence of M with input $\vec{w}$.

We say that the computation of M with $\vec{w}$ HALTS (also: CONVERGES) if there is some k s.t. the state of $C(k, M, \vec{w})$ is q_H.

Otherwise, we say it DOESN'T HALT (also: DIVERGES).

If M halts at input $\vec{w}$, we say that k is the HALTING TIME of the computation if k is least s.t. the state of $C(k, M, \vec{w})$ is q_H.

If M halts at input $\vec{w}$ with halting time k, the register content of $C(k, M, \vec{w})$ is called the register content at halting time.

Definition We say that two RM M & M' are strongly equivalent if for all $\vec{w}$ if $(C_i ; i \in \mathbb{N})$ & $(D_i ; i \in \mathbb{N})$ are the comp. of M, M' w/ input $\vec{w}$, respectively, then for each i:

① the reg. content of C_i & D_i is the same

② C_i is in the halting state $\iff$ D_i is in the halting state.

Lemma If $|Q| = |Q'|$, then for each $M = (\Sigma, Q, P)$ there is a P' s.t. (Σ, Q', P') is strongly eq. to M.

Proof. Define P' via the bij. between Q, Q', preserving halt & start. q.e.d.

Remark This means that up to strong eq., the set of states doesn't matter, only its cardinality.

Also: we can w.l.o.g. assume that state sets are disjoint.

Proposition For any fixed Σ, there are countably many RM up to strong equivalence.

Let n be the number of states and k be the upper register index.

How many instructions do we have?

$+(\ell, a, q) \longrightarrow |\Sigma| \cdot (k+1) \cdot n$

$?(\ell, a, q, q') \longrightarrow |\Sigma| \cdot (k+1) \cdot n^2$

$?(\ell, \varepsilon, q, q') \longrightarrow (k+1) \cdot n^2$

$-(\ell, q, q') \longrightarrow (k+1) \cdot n^2$

Find a nice upper bound

$I(n,k)$ s.t.

there are at most $I(n,k)$ such instructions

Thus, there are at most

$$(I(n,k))^n$$ many programs.

So the set of RM with fixed Q with $|Q| = n$ and u.r.i. $\leq k$ is finite. $R_{n,k}$

Consider

$$\bigcup_{n,k\in\mathbb{N}} R_{n,k}$$

Countable union of finite sets, so countable.

By the Lemma, this set contains a strongly eq. machine for every possible RM.

q.e.d.

Proposition (The Padding Lemma)

For every RM there are infinitely many RMs that are strongly eq. to it.

Proof. If M is given, then every computation sequence

$$C(k, M, \vec{w})$$

only uses states in Q.

Find $\hat{q} \notin Q$ and define $Q^+ := Q \cup \{\hat{q}\}$

[Note $|Q^+| = |Q| + 1$,
so $Q^+ \neq Q$.]

Then by trivial induction, the RM M^+ with $M^+ = (\Sigma, Q^+, P^+)$

where $P^+ = P \cup \{\hat{q} \longmapsto ?(0, \varepsilon, \hat{q}, \hat{q})\}$

has exactly the same computation seq. as M.

So M, M^+ are strongly equivalent.

Repeat to obtain $M^{(k)}$ with

$$|Q^{(k)}| = |Q| + k,$$ thus infinitely many diff. machines. q.e.d.

C.2 Performing operations and answering questions

We're considering partial functions. We write

$$F : X \dashrightarrow Y$$

for F is a partial function with

$$\mathrm{dom}(F) \subseteq X$$
$$\mathrm{ran}(F) \subseteq Y.$$

We also write for $x \in X$

$$F(x)\downarrow \quad :\Longleftrightarrow \quad x \in \mathrm{dom}(F)$$

"is defined"
"converges"

$$F(x)\uparrow \quad :\Longleftrightarrow \quad x \notin \mathrm{dom}(F)$$

"is not defined"
"diverges".

We can concatenate partial functions

$F : X \dashrightarrow Y$
$G : Y \dashrightarrow Z$ then $G \circ F : X \dashrightarrow Z$.

Consider a RM M as a partial function

$$F_M : \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}^{n+1}$$

where $F_M(\vec{w})\uparrow$ if the M-comp. w/ input $\vec{w}$ doesn't halt

& $F_M(\vec{w})\downarrow$ & $F_M(\vec{w}) = \vec{v}$ if the M-comp. w/ input $\vec{w}$ halts & $\vec{v}$ is the reg. content at halting time.

<u>Definition</u> Let $F : \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}^{n+1}$. We say M performs F if $F_M = F$.

<u>Example</u> Operation NEVER HALT represented by $F : \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}^{n+1}$ s.t. $\mathrm{dom}(F) = \emptyset$.

E.g., $q_S \longmapsto +(0, a, q_S)$ produces an infinite loop.

<u>Remark</u>: There are LOTS of machines that do this. In particular, if q_H does not show in P, it's going to perform F.

Example 2 ALWAYS HALT, DO NOT CHANGE INPUT

Represented by $F = id : \mathbb{W}^{k+1} \to \mathbb{W}^{k+1}$ (total)

$q_S \longmapsto ?(0, \varepsilon, q_H, q_H)$

A question with k+1 answers is a partition $\mathbb{W}^{k+1} = \bigcup_{i \leq k} A_i$ where the A_i are disjoint. (answer sets).

A RM M answers question $(A_i \mid i \leq k)$ if it has k+1 many specific answer states $\hat{q}_i$ and, upon input $\vec{w}$ it produces in a finite amount of time a configuration

$(\hat{q}_i, \vec{w}) \iff \vec{w} \in A_i$

this is the same as input!!

Example 1 Is register i empty?

Possible answers: Yes / No

$A_0 := \{\vec{w} ; w_i = \varepsilon\}$, YES

$A_1 := \{\vec{w} ; w_i \neq \varepsilon\}$ NO

$q_S \longmapsto ?(i, \varepsilon, \hat{q}_0, \hat{q}_1)$

Example 2 Does register i end with letter a?

$A_0 := \{\vec{w} ; \exists w (w_i = wa)\}$

$A_1 := W^{n+1} \setminus A_0$.

$q_S \longmapsto ?(i, a, \hat{q}_0, \hat{q}_1)$.

Proposition (The concatenation lemma)

The subroutine lemma

If M performs F & M' performs F', then there is a RM performing $F' \circ F$.

Proof. W.l.o.g., let's assume that $Q \cap Q' = \emptyset$.

Let P^* be P where all occurrences of q_H are replaced by q'_S, removing $P(q_H)$.

Define $\hat{Q} := Q \cup Q'$
$\hat{P} := P^* \cup P'$
And $\hat{M} = (\Sigma, \hat{Q}, \hat{P})$ which performs $F' \circ F$.

q.e.d.

<u>Proposition</u> (Case Distinction Lemma)

Suppose $A = (A_i ; i \leq k)$ is a question answered by M and for $i \leq k$, $F_i : W^{n+1} \dashrightarrow W^{n+1}$ is an operation performed by M_i, then

$$F(\vec{w}) := F_i(\vec{w}) \iff \vec{w} \in A_i$$

is performed by a RM.

<u>Proof.</u> Again, w.l.o.g. assume that for all i
$Q \cap Q_i = \emptyset$
$i \neq j \quad Q_i \cap Q_j = \{q_H\}$ and $P_i(q_H) = P_j(q_H)$.

Let P^* be P whose $\hat{q}_i$ is replaced by p_S^i. Then $\hat{Q} := Q \cup \bigcup_{i \leq k} Q_i$; $\hat{P} := P^* \cup \bigcup_{i \leq k} P_i$

Then $\hat{M} = (\Sigma, \hat{Q}, \hat{P})$ performs the case distinction operation.

q.e.d. :)

REMARK. Both of these Lemmas are "algorithmic" or "functional". They provide a CONCRETE RM that can be explicitly given.

Example

$$F(\vec{w}) := \begin{cases} \vec{w} & w_i \neq \varepsilon \\ \uparrow & w_i = \varepsilon \end{cases}$$

This can be performed by a RM:

Check whether reg i is empty,
if so, halt without changing anything;
if not, don't halt.

IMPORTANT REMARK

While this looks like natural language, it is fully formalized, since CHECK WHETHER REG i IS EMPTY, HALT W/O CHANGING ANYTHING, & DO NOT HALT have formal definitions as RM.