XVII

SEVENTEENTH LECTURE AUTOMATA & FORMAL LANGUAGES

MONDAY 13 NOVEMBER 2023

Some comments on §4.2

(1) The definition of performing / answering referred to the upper register index. That is not compatible with the existence of scratch space.

Fix an upper register index m and $n \leq m$. If $\vec{w} = (w_0, ..., w_n) \in \mathbb{W}^{n+1}$, we write $\vec{w}^+$ for the $m+1$-tuple $(w_0, ..., w_n, \varepsilon, ..., \varepsilon) \in \mathbb{W}^{m+1}$, i.e., the tuple $\vec{w}$ with all remaining entries filled up with the empty word. We think of the registers with the indices $n+1$ to m as *scratch space* that the machine can use to keep information while performing its computation.

Let $F : \mathbb{W}^{n+1} \dashrightarrow \mathbb{W}^{n+1}$ be any partial function. We say that a register machine M with upper register index m *performs the operation* F if for all $\vec{w} \in \mathbb{W}^{n+1}$

(i) if $F(\vec{w})\uparrow$, then M diverges on input $\vec{w}^+$ and

(ii) if $F(\vec{w})\downarrow = \vec{v}$, then M converges on input $\vec{w}^+$ with register content $\vec{v}^+$ at time of halting.

If F is a total function, we sometimes emphasise this by using the phrase "M performs the total operation F".

A *question about $n+1$-tuples with $k+1$ answers* is a partition of $\mathbb{W}^{n+1}$ into $k+1$ disjoint sets $A_0, ..., A_k$. E.g., the question "does the second register end with a?" is the partition $A_0 := \{\vec{w}; \exists v(w_2 = va)\}$ and $A_1 := \mathbb{W}^{n+1} \setminus A_0$. A register machine M with upper register index m *answers a question about $n+1$-tuples with $k+1$ answers* if it has $k+1$ designated *answer states* $\widehat{q_0}, ..., \widehat{q_k}$, and input input $\vec{w}^+ \in \mathbb{W}^{m+1}$, the computation of M with input $\vec{w}$ produces in finitely many steps a configuration $(\widehat{q_i}, \vec{w}^+)$ if and only if $\vec{w} \in A_i$.

Fixed now in the typed notes.

(2) In some of our examples, we used

REPEAT ... UNTIL

in the descriptions defining the combined machines. This does not directly follow from the subroutine lemma.

Let $Q = \{A_0, A_1\}$ be a question about n-tuples with two answers and $F : \mathbb{W}^n \dashrightarrow \mathbb{W}^n$ an operation. Define by recursion $F^0(\vec{w}) := \vec{w}$ and $F^{m+1}(\vec{w}) := F(F^m(\vec{w}))$ and

$$R_{F,Q}(\vec{w}) := \begin{cases} F^m(\vec{w}) & \text{if } m \text{ is the least number such that } F^m(\vec{w}) \in A_1 \text{ and} \\ \uparrow & \text{if there is no such number.} \end{cases}$$

The operation $R_{F,Q}$ can be described as "repeat F until the answer to Q is A_1".

Lemma 4.8 (Repeat Lemma)**.** If $Q = \{A_0, A_1\}$ be a question about n-tuples with two answers that can be answered by a register machine and $F : \mathbb{W}^n \dashrightarrow \mathbb{W}^n$ an operation performed by a register machine. Then $R_{F,Q}$ is performed by a register machine.

Proof. Let $M = (\Sigma, Q, P)$ be a register machine performing F and $M' = (\Sigma, Q', P')$ be a register machine answering Q. As before, we can assume that $Q \cap Q' = \varnothing$; let q_S, q'_S, q_H, and q'_H be the start and halt states of M and M', respectively. We define $\widehat{M} := (\Sigma, \widehat{Q}, \widehat{P})$ where $\widehat{Q} := Q \cup Q'$ and $\widehat{P}$ is $P \cup P'$ where all occurrences of $\widehat{q_0}$ (the answer state for A_0) in the instructions are replaced by q_S and all occurrences of q_H are replaced by q'_S. The start state of $\widehat{M}$ is q'_S and the halt state is $\widehat{q_1}$ (the answer state for A_1). Then $\widehat{M}$ performs the operation $R_{F,Q}$. Q.E.D.

Added the REPEAT LEMMA to the typed notes (Lemma 4.8).

RECAP Lecture XVI

Register machine M.
Partial function $f_{M,k} : \mathbb{W}^k \dashrightarrow \mathbb{W}$

$f: \mathbb{W}^k \dashrightarrow \mathbb{W}$ is **COMPUTABLE** if there is M s.t. $f = f_{M,k}$

Examples: identity, constant functions, projections

$A \subseteq \mathbb{W}^k$ (Rec/a) characteristic fn χ_A

(Rec/a) pseudocharacteristic (partial) fn ψ_A.

A is **COMPUTABLE** if χ_A is.

COMPUTABLY ENUMERABLE if ψ_A is.

C.E.

Remark A computable $\Longrightarrow$ A c.e.

Continuing with Remarks from Lecture XVI

(3) If A is computable, then so is $\mathbb{N}^k \setminus A$.

[$f : w \longmapsto \begin{cases} a & \text{if } w = \varepsilon \\ \varepsilon & \text{if } w \neq \varepsilon \end{cases}$

This fn is computable.

Check whether O is empty.

YES ↙ NO ↘

Add a to O. Halt.

Empty reg. O Halt.

Clearly $\chi_{\mathbb{N}^k \setminus A} = f \circ \chi_A$.]

Note: In the terminology of closure properties, this means that the computable set are closed under complement.

Proposition Every regular language is computable.

Proof. Let $D = (\Sigma, Q, \delta, q_0, F)$ be a det. automaton recognising $L \subseteq W$.

Define RM $M = (\Sigma, \hat{Q}, P)$ where $\hat{Q}$ has particular (disjoint) subsets Q_q for each $q \in Q$. Each of these sets has a particular initial state.

First step: Reverse reg. 0 into reg. 1.

Then start in the initial state of the set Q_{q_0}.

SUBROUTINE: if you are in the initial state of Q_q, do the following:

Read final letter of reg. 1

NON-EMPTY, say b

EMPTY

Empty register 0.

$q' := \delta(q, b)$

Go into the initial state of $Q_{q'}$.

$q \notin F$

$q \in F$

HALT.

Add a to register 0.

HALT.

This RM accepts exactly the elements of L. q.e.d.

On ES#3, you will prove that all noncontracting languages are computable.

REGULAR — Type 3
⇓
CONTEXT-FREE — Type 2
⇓
NONCONTRACTING = CONTEXT-SENSITIVE — Type 1
⇓
COMPUTABLE
⇓
C.E.

Q
Where is Type 0?

Example $\{a^u b^u c^u ; u > 0\}$ is not context-free

[Move c's to neg. 1 until none left ;
(from the back)
move b's to neg. 2 until none left ;
move a's to neg. 3 until none left.
If anything is left ⟶ NO
O/w compare lengths of neg. 1, 2, & 3.
If equal YES ; o/w NO.]

§4.4 The shortlex ordering

Let $\Sigma = \{a_0, \dots, a_n\}$ with a fixed ordering

$a_0 < a_1 < \dots < a_n$.

If $w, v \in \mathbb{W}$ and $w = b_0 \dots b_k$ and $v = c_0 \dots c_\ell$, we can define the following order relation:

$$\begin{aligned} w < v \quad :\Longleftrightarrow \quad & |w| < |v| \text{ or} \\ & |w| = |v| \text{ and } w \neq v \text{ and} \\ & \quad \text{if } i \text{ is minimal such that } b_i \neq c_i, \text{ then } b_i < c_i. \end{aligned}$$

This order relation is called the *shortlex order*. It is a total order, i.e., irreflexive (for no w, it is the case that $w < w$), transitive (if $u < v < w$, then $u < w$) and *trichotomous* (for any v and w, we either have $v < w$ or $w < v$ or $v = w$), and ε is its minimal element. We write $\mathrm{pred}_<(w) := \{v \in \mathbb{W} ; v < w\}$ for the initial segment of word smaller than w.

The shortlex ordering is a total order on $\mathbb{W}$.

Write for $w \in \mathbb{W}$

$$\mathrm{pred}_<(w) := \{v \in \mathbb{W};\ v < w\}$$

Example $\Sigma = \{0,1\}$ where $0 < 1$.

Length	Index of the word	Word	
0	0	ε	
1	1	0	
	2	1	
2	3	00	Note that the "index" is precisely $\lvert \mathrm{pred}_<(w) \rvert$.
	4	01	
	5	10	
	6	11	
3	7	000	Write #w for the "index", i.e. $\#w := \lvert \mathrm{pred}_<(w) \rvert$.
	8	001	
	9	010	
	10	011	
	11	100	
	12	101	
	13	110	
	14	111	

There is a successor function

$S : \mathbb{W} \longrightarrow \mathbb{W}$ that gives the next word in shortlex.

Theorem $(W, <) \cong (\mathbb{N}, <)$

Proof. Observe that if $w \in W$,

$\mathrm{pred}_<(w)$ is finite.

[Since all pred. are of length $\leq |w|$ and there are only finitely many such words.]

Define as on last page

$$\#w := |\mathrm{pred}_<(w)|.$$

Check that $\#$ is an order preserving bijection. q.e.d.

Proposition (P 4.15)

The set $\{(v,w);\ v<w\}$ and the function $S : w \longmapsto v$ iff $\#v = \#w + 1$.

SUCCESSOR

are computable.

Proof Assume v is in reg. 0; w is in reg. 1. Reverse them.

Check their length (Lecture XVI, Ex. 17). If v shorter $\longrightarrow$ YES

If w shorter $\longrightarrow$ NO

If $|v| = |w|$, remove letter from both one by one and compare. If equal, continue.

If different and $v_i < w_i \longrightarrow$ YES; $w_i < v_i \longrightarrow$ NO

If difference found $\longrightarrow$ NO.

Now the successor function: $\Sigma : a_0 < a_1 < \ldots < a_n$

- Move all a_n's from the back of word into a scratch register until you hit a_k with $k \neq n$

Change that to a_{k+1}.

For each a_n on scratch register add one a_0 back.

If you don't find such an a_k, add as many a_0's as are in the scratch register and one extra.

q.e.d.

§ 4.5 CHURCH'S RECURSIVE FUNCTIONS

4.5 Church's recursive functions

The following operations on partial functions were considered by Alonzo Church (1903–1995). The functions

$$\pi_{k,i} : \mathbb{W}^k \to \mathbb{W} : \vec{w} \to w_i \text{ (projection functions)}$$
$$c_{k,\varepsilon} : \mathbb{W}^k \to \mathbb{W} : \vec{w} \to \varepsilon \text{ (constant functions)}$$
$$s : \mathbb{W} \to \mathbb{W} : w \mapsto v \quad \text{(where } \#(v) = \#(w) + 1\text{; the successor function).}$$

are called *basic functions*. We have already proved that all basic functions are computable.

Suppose $f : \mathbb{W}^m \dashrightarrow \mathbb{W}$ and $g_1, \ldots, g_m : \mathbb{W}^k \dashrightarrow \mathbb{W}$ are partial functions, then the partial function h defined by

$$h(\vec{w}) := f(g_1(\vec{w}), \ldots, g_m(\vec{w}))$$

is called the *composition of f with $(g_1, \ldots, g_m)$*. The notational convention used for operations applies here as well: if any term on the right hand side is undefined, then so is the left hand side.

Suppose $f : \mathbb{W}^k \dashrightarrow \mathbb{W}$ and $g : \mathbb{W}^{k+2} \dashrightarrow \mathbb{W}$ are partial functions, then the function h defined by the recursion equations

$$h(\vec{w}, \varepsilon) = f(\vec{w}) \text{ and}$$
$$h(\vec{w}, s(v)) = g(\vec{w}, v, h(\vec{w}, v))$$

is called the *recursion result of f and g*.

Suppose $f : \mathbb{W}^{k+1} \dashrightarrow \mathbb{W}$ is a partial function, then the partial function h defined by

$$h(\vec{w}) := \begin{cases} v & \text{if for all } u \leq v, \text{ we have that } f(u)\downarrow \text{ and} \\ & \quad v \text{ is } <\text{-minimal such that } f(\vec{w}, v) = \varepsilon \text{ or} \\ \uparrow & \text{if for all } v,\ f(\vec{w}, v) \neq \varepsilon \end{cases}$$

is called the *minimisation result of f*.

We say that a class $\mathcal{C}$ of partial functions is closed under composition, recursion, or minimisation if, whenever $f, g, g_1, \ldots, g_m$ are in $\mathcal{C}$, then the composition of f with $(g_1, \ldots, g_m)$, the recursion result of f and g, or the minimisation result of f, respectively, are in $\mathcal{C}$.

MINIMISATION NEXT TIME

Alonzo Church

Alonzo Church (1903–1995)

Born	June 14, 1903 Washington, D.C., US
Died	August 11, 1995 (aged 92) Hudson, Ohio, US
Citizenship	United States
Alma mater	Princeton University
Known for	Lambda calculus Simply typed lambda calculus Church encoding Church's theorem Church–Kleene ordinal Church–Turing thesis Frege–Church ontology Church–Rosser theorem Intensional logic

Church's recursive fns are defined by closure operations

BASIC FUNCTIONS: projections, constant, successor.

COMPOSITION: concatenation

RECURSION: $h(\vec{w}, \varepsilon) = f(\vec{w})$
$h(\vec{w}, s(v)) = g(\vec{w}, v, h(\vec{w}, v))$