Hong Jia-wei

Computation: Computability, Similarity and Duality

Hong Jia-wei
Beijing Computer Institute, China

Computation: Computability, Similarity and Duality

Pitman, London

John Wiley & Sons, Inc., New York, Toronto

PITMAN PUBLISHING LIMITED
128 Long Acre, London WC2E 9AN

A Longman Group Company

© Hong Jia-wei 1986

First published 1986

Available in the Western Hemisphere from
John Wiley & Sons, Inc.
605 Third Avenue, New York, NY 10158

ISSN 0268-7534

British Library Cataloguing in Publication Data

Hong, Jia-wei
Computation: computability, similarity
and duality.
1. Computer arithmetic
I. Title
519.4'028'5 QA76.9.C62

ISBN 0-273-08720-7

Library of Congress Cataloging in Publication Data

ISBN 0-470-20387-0

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise, without the prior written permission of the publishers. This book may not be lent, resold, hired out or otherwise disposed of by way of trade in any form of binding or cover other than that in which it is published, without the prior consent of the publishers.

Reproduced and printed by photolithography
in Great Britain by Biddles Ltd, Guildford

Preface

Computation is one of the oldest as well as one of the newest topics. Among the vast amounts of literature about computation, this book will find its own way to unify computational models and develop Turing-Church's thesis further.

This book composed of three parts. Part 1 (Chapters 1, 2 and 3) is a brief self-contained review, which discusses finite automata and classical computability theory. The material selected is as basic as possible, and the proof as simple as possible. The reader who is familiar with these materials can go to Part 2 directly.

Part 2 consists of Chapters 4, 5, 6 and 7. In Chapter 4, multitape Turing machines (TM) and computational complexities are introduced to the reader. In Chapters 5 and 6, random access machines (RAM) and vector machines (VM) are discussed, and their similarity with TM is proved. These three models are developed in full so that senior students and junior graduate students can understand the similarity theory for deterministic computational models. Chapter 7 provides three more parallel computational models which are proved to be similar to TM. Another four models and further developments are given in the exercises and remarks.

Part 3 consists of Chapters 8, 9 and 10. Chapters 8 and 9 are intended to unify various computational types. Especially non-deterministic, alternating, majority and random types, and their relations are considered. In Chapter 10, the duality between parallel time and space is discussed. This part can be treated as a separate text for further reading by graduate students.

This book is self-contained, but readers are recommended to complete the exercises in order to understand the text better.

The outline for this book was reported as a lecture at the 21st FOCS Symposium in 1980, entitled "On Similarity and Duality of Computation", based mainly on work I had done when I was visiting Toronto. Many valuable ideas were suggested by my friends A. Borodin, S. Cook, P. Dymond and C. Rackoff. In 1982, I wrote a lecture of 11 chapters for a theoretical seminar.

in Beijing Computer Institute and, in 1983, wrote a clearer version with the great help of S.W. Tang, for a summer school in Beijing. In 1985, I wrote it once again for *Research Notes in Theoretical Computer Science*, with continual encouragement from R. Book and A. Rosenberg, and helped by S.W. Tang, H.A. Wang and X.F. Liu. I would like to thank them all faithfully.

Hong, Jia-wei

Beijing Computer Institute,
China

试读结束：需要全本请在线购买：www.ertongbook.com

Contents

Preliminaries

I. NOTATION

In this book, sets are denoted by capital letters, while its elements are denoted by lower-case letters. The total number of elements in set A is denoted by $|A|$. $\emptyset$ is the null set.

The union, intersection and difference of two sets are represented by $\cup$, $\cap$ and - respectively.

$$A \cup B = \{x | x \in A \text{ or } x \in B\},$$

$$A \cap B = \{x | x \in A \text{ and } x \in B\},$$

$$A - B = \{x | x \in A \text{ and } x \notin B\}.$$

Notation $A \subseteq B$ means that A is a subset of B, while $A \subset B$ means $A \subseteq B$ and $B-A \neq \emptyset$.

The Cartesian product of A and B are defined by

$$A \times B = \{(a,b) | a \in A, b \in B\}.$$

Define $A^1 = A$, $A^{n+1} = A^n \times A$. Use 2^A to represent the power set of A:

$$2^A = \{B | B \subseteq A\}.$$

Use A^B to represent the set of all mappings from B to A.

A relation R on set A is a subset of $A \times A$. For two elements, $a,b \in A$, that a and b have relation R means $(a,b) \in R$, denoted by aRb.

The positive closure R^+ of a relation R is a relation defined by: aR^+b if and only if there are $a = a_1, a_2, \ldots, a_n = b$ $(n \geq 2)$ such that $a_i R a_{i+1}$ $(i = 1,2,\ldots,n-1)$. The closure R^* of relation R is defined by: aR^*b if $a = b$ or aR^+b.

For any real number x, $\lfloor x \rfloor$ is the unique integer satisfying $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$. $\lceil x \rceil$ is the unique integer satisfying $\lceil x \rceil - 1 < x \leq \lceil x \rceil$. The meaning of $[x]$ is the same as $\lfloor x \rfloor$.

II. ALPHABET AND LANGUAGE

An alphabet is a finite set, whose elements are called the **symbols**. Suppose that I is an alphabet. A string of a finite number (including 0) of symbols from I is a word over I. The word of length 0 is called the **null word**, denoted by Λ. The null word Λ is not a symbol in an alphabet but a string having no symbol, a word (empty word) over any alphabet. The set of all words in alphabet I will be denoted by I^*. For example, if $I = \{a,b\}$, then $I^* = \{\Lambda,a,b,aa,ab,ba,bb,aaa,\ldots\}$.

Suppose that $x = a_1a_2\ldots a_n$ and $y = b_1b_2\ldots b_m$ are two words in alphabet I. Then the word $w = xy = a_1a_2\ldots a_n\, b_1b_2\ldots b_m$ is called the concatenation of x and y. Obviously, we have

(1) $(xy)z = x(yz) \quad x,y,z \in I^*$

(2) $\Lambda x = x\Lambda = x \quad x \in I^*$.

If $xy = z$, then x is a prefix of z, and y is a suffix of z. Further, if we have $y \neq \Lambda$ (or $x \neq \Lambda$), then x (or y) is a proper prefix (proper suffix).

If $xyz = w$, then y is a subword of w. If we have $y \neq \Lambda$ and $y \neq w$ then y is a proper subword of w.

Suppose that $x = a_1a_2\ldots a_n$ $(a_i \in I,\ i = 1,2,\ldots,n)$, then $a_n\ldots a_2a_1$ is the reversed word of x, denoted by x^r.

Let x be a word, $n \geq 0$. Define

$$x^0 = \Lambda$$

$$x^{n+1} = x^n \cdot x.$$

Obviously we have

(1) $x^n \cdot x^m = x^{n+m} \quad (x \in I^*,\ n,m \geq 0)$

(2) $(x^n)^m = x^{nm} \quad (x \in I^*,\ n,m \geq 0)$.

A subset of I^* is called a language over I.

Suppose that L_1 and L_2 are two languages over I. Define language

$$L = L_1 \cdot L_2 = \{xy \mid x \in L_1,\ y \in L_2\}$$

to be the concatenation of L_1 and L_2. Obviously we have

(1) $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$.

(2) $\{\Lambda\} \cdot L = L \cdot \{\Lambda\} = L$.

(3) $\emptyset \cdot L = L \cdot \emptyset = \emptyset$

(4) $(L_1 \cup L_2) \cdot L = L_1 \cdot L \cup L_2 \cdot L$

$L \cdot (L_1 \cup L_2) = L \cdot L_1 \cup L \cdot L_2$.

Thus $\{\Lambda\}$ and $\emptyset$ are essentially different. Furthermore, they are different from $\sqcup$, the blank symbol.

Suppose that L is a language over I, $n \geq 0$. Define

$$L^0 = \{\Lambda\}$$

$$L^{n+1} = L^n \cdot L$$

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

$$L^+ = \bigcup_{n=1}^{\infty} L^n.$$

L^* is the Kleene closure of L. L^+ is the positive Kleene closure of L. Obviously we have

$$(L^*)^* = L^*$$

$$L^+ = L \cdot L^* = L^* \cdot L.$$

Therefore $\emptyset^* = \{\Lambda\}$ but $\emptyset^+ = \emptyset$.

In the following we denote $\{w\}^*$ as w^*, and $\{w\}^+$ as w^+.

III. GRAPH

Suppose that V,E are finite sets and for every $e \in E$ there corresponds a unique $(u,v) \in V \times V$. Then $G = (V,E)$ is a directed graph. The elements in V are called the vertices. The elements in E are called the edges. If $(u,v) \in V \times V$ corresponds to $e \in E$, then u is the start point of e, and v is the end point of e.

The number of edges that have v as start (end) point is the fan-out (fan-in) number. A word $P = e_1 \ldots e_n$ over E satisfying that the end point of

试读结束：需要全本请在线购买： www.ertongbook.com

e_i is the start point of $e_{i+1}(i = 1,2,\ldots,n-1)$, is called a path in the graph. The path is of length n. A path of length 0 is called a null path. Thus for every vertex v there is a null path from v to v. A non-null path having the same start point and end point is a cycle.

Suppose that G is a directed graph, I is a set, f is a mapping from E to I. Then (G,f) is an edge-assignment directed graph. f is called the assignment mapping. For $e \in E$, f(e) is the assignment of e. Sometimes, G can be referred to as an I assignment directed graph. The assignment of a path $P = e_1e_2\ldots e_n$ is defined to be $f(P) = f(e_1)f(e_2)\ldots f(e_n)$, which is a word over I.

Reversing the direction of all edges in a directed graph G, a new directed graph is obtained, the reversed graph G^r. If G is an I assignment graph, every edge in G^r keeps the original assignment. Obviously, P is a path in G if P^r is a path in G^r.

For each vertex in a directed graph, the set

$$L(v) = \{u \mid (v,u) \in E\}$$

is called the adjacency list of G.

For $G = (V,E)$, $V = \{v_1,v_2,\ldots,v_n\}$, the following matrix

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

where

$$a_{ij} = \begin{cases} 1, & (v_i,v_j) \in E \\ 0, & (v_i,v_j) \notin E \end{cases}$$

is called the adjacency matrix of G.

The edge set of a directed graph is in fact a relation of V. If the relationship is symmetric, i.e., $(u,v) \in E$ iff $(v,u) \in E$, then the graph is an undirected graph.

Part one
Finite automata and computability
1 Finite automata

§1. DETERMINISTIC FINITE AUTOMATA

A deterministic finite automaton is a mechanism with a finite number of inner states. When it receives an input symbol it will respond with an output symbol and change to another state according to the current state and input symbol.

DEFINITION 1.1 A determinisitc finite automaton (DFA) is a 7-tuple:

$$M = (Q,I,U,\delta,\sigma,q_0,F)$$

where

Q is a finite set of inner states;
I is a finite set, the input alphabet;
U is a finite set, the output alphabet;
δ is a mapping from $Q \times I$ to Q, the state transition function;
σ is a mapping from $Q \times I$ to U, the output function;
$q_0 \in Q$, the initial state;
$F \subseteq Q$, the set of final states.

Informally, a DFA has an FC, an input tape and an output tape. The FC, controlling a read head and a write head, is always in some state $q \in Q$ at any moment. The input tape and output tape are divided into many squares, each can store one symbol from I or U.

Initially, the FC is in the initial state q_0. The input, stored on the input tape, is a string $(a_1a_2...a_n)$ of symbols in I where a_1 is the first input symbol; an empty word is on the output tape, i.e. all the squares on output tape are filled with blanks. The read head is pointing to the left-most input symbol. According to the current inner state q_0 and the symbol a_1 scanned by the read head, the FC controls the write tape head to output a symbol $b_1 = \sigma(q_0,a_1)$ in the scanned square, enters a new state $q_1 = \delta(q_0,a_1)$ and the read/write heads automatically move one square to the right. Then,

the DFA repeats the above process until all the input symbols have been treated. Finally, the FC enters a state q_n and produces an output word $b_1b_2\ldots b_n$ where

$$b_{i+1} = \sigma(q_i,a_{i+1})$$

$$q_{i+1} = \delta(q_i,a_{i+1}) \quad i = 0,1,2,\ldots,n-1.$$

For convenience, we extend the domain of σ and δ from $Q \times I$ to $Q \times I^*$:

$$\delta(q,\Lambda) = q,$$

$$\delta(q,wa) = \delta(\delta(q,w),a),$$

$$\sigma(q,\Lambda) = \Lambda,$$

$$\sigma(q,wa) = \sigma(q,w)\sigma(\delta(q,w),a), \text{ where } q \in Q,\ w \in I^*,\ a \in I.$$

For any given $w \in I^*$ if $\delta(q_0,w) \in F$, we say that the DFA accepts the input word w; otherwise, the DFA rejects the input word w.

For a simple DFA, we have an intuitive expression, the state-transition diagram. The diagram is a directed graph whose vertices correspond to the states of the DFA. For $q \in Q$, $a \in I$, if $\delta(q,a) = q'$, $\sigma(q,a) = b$, then there is an arc labelled a(b) from state q to state q' in the transition diagram. The initial state is generally indicated by an arrow.

When a DFA is used as a transducer, we often set $F = \phi$ (or $F = Q$). Thus, as a transducer a DFA can be described by a 6-tuple $(Q,I,U,\delta,\sigma,q_0)$.

The language accepted by M is defined by

$$L(M) = \{w \in I^* | \delta(q_0,w) \in F\}.$$

When a DFA is used as an accepter, we usually only write

$$M = (Q,I,\delta,q_0,F).$$

Then the DFA is a 5-tuple, and the arcs of its state transition diagram are labelled only by symbols from I.

§2. NON-DETERMINISTIC FINITE AUTOMATA

In the previous section, the language accepted by a DFA is the set

$\{w \in I^* | \text{there is a path from } q_0 \text{ to some } q \in F \text{ whose assignment is } w\}$. (2.1)

For each $q \in Q$ and each $a \in I$, $\delta(q,a)$ is unique.

Now we reduce the limits so that for each $q \in Q$ and each $a \in I$ there may be a finite number, perhaps zero, of arcs labelled a out from state q. We also specify one initial vertex q_0 and one accepting vertex set F, and define the language this graph accepts by (2.1). Thus, we get a new kind of language accepter.

DEFINITION 2.1 A non-deterministic finite automaton (NDFA) is a 5-tuple, $M = (Q,I,\delta,q_0,F)$, where Q,I,F and q_0 have the same meaning as for a DFA, but δ is a mapping from $Q \times I$ to 2^Q.

The function δ can be extended to a mapping from $Q \times I^*$ to 2^Q:

$$\delta(q,\Lambda) = \{q\},$$

$$\delta(q,wa) = \bigcup_{p \in \delta(q,w)} \delta(p,a) \quad (q \in Q,\ w \in I^*,\ a \in I),$$

and also can be extended to arguments in $2^Q \times I^*$ as follows:

$$\delta(S,a) = \bigcup_{q \in S} \delta(q,a) \quad (S \in 2^Q,\ a \in I) \tag{2.2}$$

$$\delta(S,\Lambda) = S,$$

$$\delta(S,wa) = \delta(\delta(S,w),a) \quad (S \in 2^Q,\ w \in I^*,\ a \in I).$$

The language accepted by M is:

$$L(M) = \{w \in I^* | \delta(q_0,w) \cap F \neq \emptyset\}$$

THEOREM 2.1 For each NDFA with k states, there exists an equivalent DFA with 2^k states. By 'equivalent', it is meant that they both accept the same language.

Proof: let $M = (Q,I,\delta,q_0,F)$ be an NDFA. Construct a DFA as follows:

$$M' = (2^Q,I,\delta',\{q_0\},F'),$$

where

$$F' = \{S \in 2^Q \mid S \cap F \neq \emptyset\},$$

$$\delta'(S,a) = \bigcup_{q \in S} \delta(q,a).$$

According to formula (2.2), the function δ' is exactly the extension of the function δ, that is,

$$\delta'(S,a) = \delta(S,a) \quad (S \in 2^Q,\ a \in I).$$

But

$$w \in L(M') \leftrightarrow \delta'(\{q_0\},w) \in F' \leftrightarrow \delta(\{q_0\},w) \in F' \leftrightarrow \delta(q_0,w) \in F' \leftrightarrow \delta(q_0,w) \cap F \neq \emptyset \leftrightarrow w \in L(M).$$

Thus $L(M') = L(M)$.

Obviously, M' has 2^k states, where $k = |Q|$.

The model of the non-deterministic finite automaton can be extended to allow transitions on the empty input Λ.

DEFINITION 2.2 In Definition 2.1, if the function δ is a mapping from $Q \times (I \cup \{\Lambda\})$ to 2^Q instead of a mapping from $Q \times I$ to 2^Q, then we call M a non-deterministic finite automaton with Λ-moves, an NDFA with Λ-moves for short.

THEOREM 2.2 For each NDFA with Λ-moves, there is an equivalent NDFA without Λ-moves.

Proof: let $M = (Q,I,\delta,q_0,F)$ be an NDFA with Λ-moves. For each $q \in Q$, we define the Λ-CLOSURE of q as follows:

$$C_\Lambda(q) = \{p \in Q \mid \text{there is a path with assignment } \Lambda \text{ from q to p}\}.$$

For each $S \subseteq Q$, define the Λ-CLOSURE of S as

$$C_\Lambda(S) = \bigcup_{q \in S} C_\Lambda(q).$$

Now define the map δ' from $Q \times I$ to 2^Q as follows

For each $q \in Q$ and $a \in I$, $\delta'(q,a) = \delta(C_\Lambda(q),a)$.

Obviously,

$p \in \delta'(q,a) \leftrightarrow$ there is a path with a sequence of assignments Λ and finally an assignment a from q to p in M. (2.3)

Again, define

$$F' = \{q | C_\Lambda(q) \cap F \neq \phi\} \tag{2.4}$$

$$M' = (Q,I,\delta',q_0,F').$$

Thus M' is an NDFA, which accepts L(M).

§3 REGULAR LANGUAGE AND REGULAR EXPRESSION

DEFINITION 3.1 Let L be a language over the alphabet I. If L can be expressed with $\emptyset$, $\{\Lambda\}$, and $\{a\}(a \in I)$ through a finite number of operations of union $\cup$, concatenation, and Kleene closure *, ther we call the language L a regular language (regular set).

DEFINITION 3.2 Let I be an alphabet not containing the symbols (,), $\emptyset$,Λ, +, $\cdot$, and *.

(1) $\emptyset$ is a regular expression, denoting the language $\emptyset$. Λ is a regular expression, denoting the language $\{\Lambda\}$. For each $a \in I$, a is a regular expression, denoting the language $\{a\}$.

(2) If r and s are regular expressions denoting the languages R and S, respectively, then $(r+s)$, $(r\cdot s)$, and r^* are regular expressions denoting the languages $R \cup S$, $R \cdot S$, and R^*, respectively.

(3) Nothing else is a regular expression.

When no confusion is possible, some parentheses can be omitted. From here on, the language described by a regular expression r is denoted by L(r). Given a regular expression r, we use r^+ instead of $r \cdot r'$. Obviously, $L(r^+) = L(r)^+$. The languages described by the regular expressions are regular, and vice versa.

试读结束：需要全本请在线购买： www.ertongbook.com