Introduction
Complete tables (TC)
Abbreviated tables (TA)
Linear alphabetic descriptions (TL_a)
Linear digital descriptions (TL_n)
Conclusion

Turing Languages: Theory and Practice

Baptiste Mélès

CNRS, Archives Henri-Poincaré—PReST (UMR 7117), Université de Lorraine

Autumn workshop II:
Programming Languages and Notations,
Bertinoro, Italy,
1-2 October 2019

Outline

1. Introduction
2. Complete tables (TC)
3. Abbreviated tables (TA)
4. Linear alphabetic descriptions (TL_a)
5. Linear digital descriptions (TL_n)
6. Conclusion

1 Introduction
2 Complete tables (TC)
3 Abbreviated tables (TA)
4 Linear alphabetic descriptions (TL_a)
5 Linear digital descriptions (TL_n)
6 Conclusion

What is a Turing machine?

- What is a Turing machine?
- One simple answer (Turing 1936, section 1): a machine
  - going through different states (*m*-configurations),
  - with a tape composed of successive squares,
  - able to scan one symbol at a time,
  - able to write down one symbol on the current square,
  - able to shift the tape to left or to right,
  - such that the "behaviour" (writing + shifting + new *m*-configuration) depends on the "configuration" (current *m*-configuration + scanned symbol).

How to describe a Turing machine?

- Now, how to describe a particular Turing machine?
- Multiple answers
  1. in a table — which kind?

This kind?

Configuration		Behaviour		
m-conf.	scan	writing	shifting	final *m*-conf.
b	None	P0	R	c
c	None	E	R	e
e	None	P1	R	f
f	None	E	R	b

This kind?

Configuration		Behaviour	
m-conf.	symbol	operations	final *m*-conf.
b	None	P0, R	c
c	None	R	e
e	None	P1, R	f
f	None	R	b

This kind?

Configuration		Behaviour	
m-conf.	symbol	operations	final *m*-conf.
b	None	P0, R, R	e
e	None	P1, R, R	b

This kind?

m-conf.	symbol	operations	final *m*-conf.
b	None	P0, R	b
	0	R, R, P1	b
	1	R, R, P0	b

This kind?

Configuration		Behaviour	
m-conf.	symbol	operations	final *m*-conf.
b		P0	b(1)
b(α)	β	R, R, Pα	b(β)

etc.

How to describe a Turing machine?

- How to describe a particular Turing machine?
- Multiple answers
  1. in a table — which kind?
  2. in an alphabetical sequence? e.g ";DADDCR-DAA;DAADDRDAAA;DAAADDCCRDAAAA;DAAAADDRDA"

How to describe a Turing machine?

- How to describe a particular Turing machine?
- Multiple answers
  1. in a table — which kind?
  2. in an alphabetical sequence? e.g ";DADDCR-DAA;DAADDRDAAA;DAAADDCCRDAAAA;DAAAADDRDA"
  3. in an digital sequence? e.g "731332531173113353111731113322531111731111133531"

Analysis of Turing languages

- There are many ways of describing the functioning of a given machine.
- When describing his machines, Turing constantly switches from one language to another: different syntaxes for different users and different uses.
- But these languages are all defined implicitly and informally.

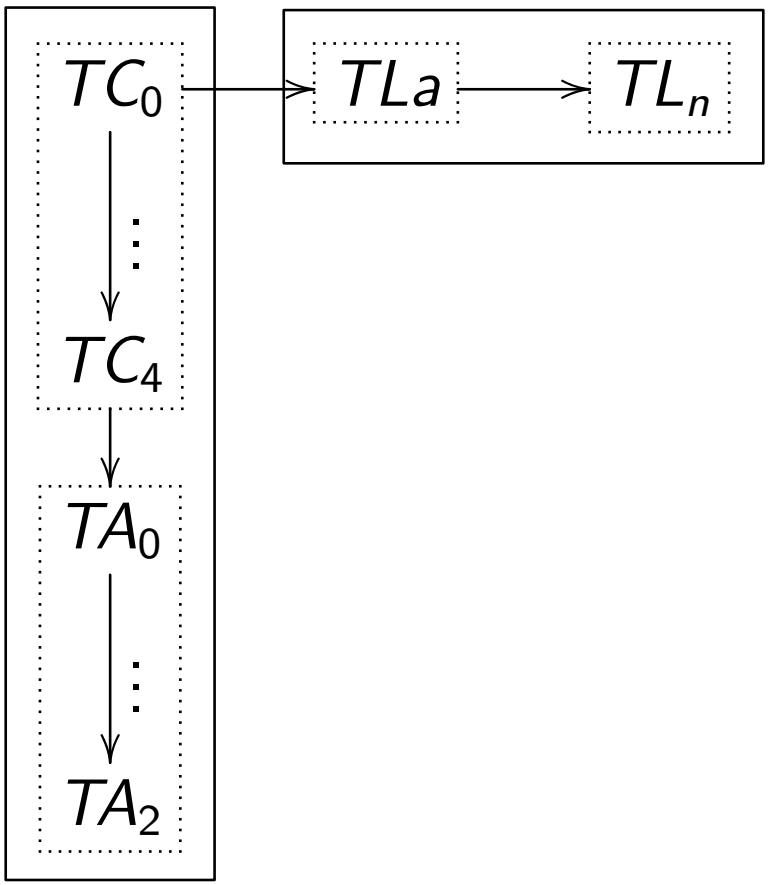
Some previous analyses of Turing languages

- Donald Knuth and Luis Trabb Pardo 1976/1980, "The Early Development of Programming Languages", Stanford U., Comp. Sci. Dept. (Aug. 1976); in N. Metropolis, J. Howlett, G.-C. Rota (eds.), *A History of Computing in the Twentieth Century*, San Diego (1980).
- Jack Copeland 2004, "Computable numbers: a guide", *The Essential Turing*, pp. 5–57.
- (Baptiste Mélès 2008: `TMPL`, a Perl implementation.)
- Mark Priestley 2011, *A Science of Operations*, ch. 4 and App. A.
- Liesbeth De Mol 2018, "Turing Machines", *The Stanford Encyclopedia of Philosophy*.

Analysis of Turing languages

- Let us describe all these languages as if they were programming languages:
  1. formal definition;
  2. analysis of examples;
  3. analysis of uses;
  4. some comparisons with concepts of modern programming.
- I will show some discrepancies between 1, 2 and 3.

Typology



Typology

1. Complete/Abbreviated tables ($TC_{0,\ 1,\ 2,\ 3,\ 4}$, $TA_{0,\ 1,\ 2}$)
   - explicative languages
   - intended for human readers and
   - allowing them to understand complex machines.
2. Standard description (TLa)
   - a computing language
   - intended for machines and
   - allowing them to handle machines as input/output.
3. Descriptive number (TLn)
   - an arithmetical language
   - intended for the mathematician and
   - allowing one to handle the cardinal and ordinal properties of computable numbers.

1 Introduction

2 Complete tables (TC)

3 Abbreviated tables (TA)

4 Linear alphabetic descriptions (TL_a)

5 Linear digital descriptions (TL_n)

6 Conclusion

Introduction
TC_0: full and faithful description of the machine
TC_1: blind lines and double operations
TC_2: multiple shifting
TC_3: multiple operations
TC_4: classes of scanned symbols
Conclusion

Two tabular languages

- Two tabular languages:
  1. complete tables: TC_0, TC_1, TC_2, TC_3, TC_4;
  2. abbreviated tables: TA_0, TA_1, TA_2.

Syntax weakening in complete tables

- "Complete tables" are a series of languages
  - TC_0 full and faithful description of the machine's behaviour
  - TC_1 syntax weakening making tables shorter
  - TC_2 some more syntax weakening
  - TC_3 still more syntax weakening
  - TC_4 even more syntax weakening making tables shorter and shorter
- The constant if the explicit naming of *m*-configurations.

TC_0 as reference language

- TC_0 is both
  - the base language for machine descriptions;
  - designed for human readers.

A ghost language

- But this reference language is nowhere to find!
  - no formal definition;
  - no example.
- However, it is sometimes alluded to and can be reconstructed from three convergent sources:
  1. the definition of Turing machines (section 1);
  2. by ignoring further simplifications (section 3);
  3. by transposing what Turing later calls "standard descriptions" (section 5).

Structure of the language

In order to define this language, one must know how the machine works:

> *The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $S(r)$. This pair $q_n, S(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine.*

Introduction
TC_0: full and faithful description of the machine
TC_1: blind lines and double operations
TC_2: multiple shifting
TC_3: multiple operations
TC_4: classes of scanned symbols
Conclusion

Configuration and behaviour

1. Configuration
   1. *m*-configuration
   2. scanned symbol
2. Behaviour
   1. writing
   2. shifting
   3. final *m*-configuration

Configuration 1/2: *m*-configuration

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q_1, q_2, ... q_k which will be called "m-configurations".

Configuration 2/2: scanned symbol

The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $S(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". [...]

Behaviour 1/3: writing

In some of the configurations in which the scanned square is blank (i.e. bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol.

Behaviour 2/3: shifting

The machine may also change the square which is being scanned, but only by shifting it one place to right or left.

Behaviour 3/3: final *m*-configuration

In addition to any of these operations the m-configuration may be changed.

Formal description 1/3

Configuration		Behaviour		
m-conf.	scan	writing	shifting	final *m*-conf.
Q	S	P	D	Q

- Q finite set of letters representing the *m*-configurations (in particular "*b*" for the starting *m*-configuration);
- S finite set of expressions (most of them autonymous) denoting readable and writable symbols ("*0*" for the symbol "0", "*1*" for "1", etc.; "None" for the empty character);

Formal description 2/3

Configuration		Behaviour		
m-conf.	scan	writing	shifting	final *m*-conf.
Q	S	P	D	Q

- P (finite) set of expressions modifying the tape: "E" (erasement) and, for each element x of $S \setminus \{None\}$, "Px" ("$P0$" for "write down "0"", "$P1$" for "write down "1"", etc.);
- D (finite) set of symbols representing shifts of the tape, defined by enumeration: "L" to shift it to left, "R" to right, "N" for the null shifting.

Formal description 3/3

Configuration		Behaviour		
m-conf.	scan	writing	shifting	final *m*-conf.
Q	S	P	D	Q

L_0 (finite) set $Q \times S \times P \times D \times Q$ of well-formed lines;

T_0 (finite) set of (finite) sequences (finies) of elements of L_0 whose first two columns are unique.

Example

TC_0 description of the machine I, which writes down the sequence "010101..." (reconstitution based on section 3, p. 233 and section 5, p. 241):

Configuration		Behaviour		
m-conf.	scan	writing	shifting	final *m*-conf.
b	None	P0	R	c
c	None	E	R	e
e	None	P1	R	f
f	None	E	R	b

Use: A description language for human readers

- Admittedly, TC_0 describes directly the functioning of the machine.
- But it is designed for human readers:
  - mnemonic names: *b* for "begin", *E* for "erase", *P* for "print", *L* for "left", *R* for "right";
  - English or autonymous symbol names ("None", "0", "1");
  - alphabetic order: *c*, *e*, *f*;

Use: A description language for human readers

TC_0 is designed for human readers to understand the machine's behaviour.

1. it describes the behaviour:

   > *A table does not describe the physical structure of machine [...], but its behaviour, or the sequence of basic operations that it will carry out. (Priestley 2011, p. 79)*

2. it describes the behaviour, i.e plays no causal role in it: it is not a program.

Use: A description language for human readers

To some extent, TC_0 could be seen as an "assembly language": human-readable description of a low-level program.

```
0 LOAD 9
1 SUB 10
2 JEZ 8
3 STO 9
4 LOAD 11
5 ADD# 1
6 STO 11
7 JMP 0
8 STOP
9 14
```

Use: A description language for human readers

- However, unlike assembly language, it is not supposed to be compiled (nor interpreted) by a machine.
- It is not a programming language (intended to be executed by a machine) but a *description language*.

TC_0 as a crossroad

- What is proper to TC_0 is to be both faithful to the machine and humanly readable;
- all further languages will have to make a choice between
  - human readability: TC_1, TC_2, TC_3, TC_4 and then TA_0, TA_1 and TA_2 are more and more explicative;
  - faithfulness to the machine: TL_a (standard descriptions), TL_n (descriptive numbers).

Features of TC_1

In section 3, Turing simplifies the syntax (double operations):

- he merges the columns "writing" and "shifting";
- he eliminates trivial operations like "E" when the square is empty and "N".

Example (section 3, p. 233/55):

Configuration		Behaviour	
m-conf.	symbol	operations	final *m*-conf.
b	None	P0, R	c
c	None	R	e
e	None	P1, R	f
f	None	R	b

Abstracted scanned symbols

Most of all, Turing allows one to empty the column of scanned symbols (blind lines).

> *When the second column is left blank, it is understood that the behaviour of the third and fourth columns applies for any symbol and for no symbol.*

m-conf.	symbol	operations	final *m*-conf.
p	∅	P0, R	q

Example

m-conf.	symbol	operations	final *m*-conf.
p		P0, R	q

also means

m-conf.	symbol	operations	final *m*-conf.
p	None	P0, R	q
p	0	P0, R	q
p	1	P0, R	q

Warning: do not confuse blank square with the "None" symbol!

Formal description (1/3)

m-conf.	symbol	operations	final m-conf.
Q	S_1	O_1	Q

S_1 (finite) set of symbols of symbols, defined by enumeration:

1. for each $s \in S$, s belongs to S_1;
2. the empty symbol ε (which denotes any symbol) belongs to S_1;

D_1 (finite) set of non-trivial shifting symbols, defined by enumeration: "L" for a shift to left, "R" for a shift to right;

Formal description (2/3)

m-conf.	symbol	operations	final m-conf.
Q	S_1	O_1	Q

O_1 (finite) set of operations, which replaces the cartesian product $P \times D$ and is defined by enumeration:

1. for each $p \in P$ and $d \in D_1$, "p, d" belongs to O_1;
2. for each $p \in P$, "p" belongs to O_1 (which replaces "p, N");
3. for each $d \in D_1$, "d" belongs to O_1;
4. the empty symbol ε belongs to O_1.

Formal description (3/3)

m-conf.	symbol	operations	final m-conf.
Q	S_1	O_1	Q

L_1 (finite) set $Q \times S_1 \times O_1 \times Q$ of well-formed lines;

T_1 (finite) set of (finite) sequences of elements of L_1 whose first two columns are unique.

Why use TC_1?

m-conf.	symbol	operations	final m-conf.
Q	S_1	O_1	Q

- Tables are smaller:
  - 1 column less;
  - many lines less (division by $\sim \#S$ w.r.t. TC_0 in the best case).
- Consequence:
  - tables are easier to read;
  - there are several ways of describing the same machine.
- TC_1 also shows a symmetry between configuration and behaviour:
  - input: m-configuration + state of the band;

Description

- TC_2 is a new simplification of the language (multiple shifting):

  If (contrary to the description in § 1) we allow the letters L, R to appear more than once in the operations column we can simplify the table considerably. (section 3, p. 234/55)

Example (reconstitution):

m-conf.	symbol	operations	final *m*-conf.
b	None	P0, R, R	e
e	None	P1, R, R	b

Formal description (1/2)

m-conf.	symbol	operations	final *m*-conf.
Q	S_1	O_2	Q

Ω_2 (infinite) set of TC_2 operations, defined by induction (not enumeration!):

1. ε belongs to Ω_2;
2. for each $p \in P$, "p" belongs to Ω_2;
3. for each $d \in D_1$, "d" belongs to Ω_2;
4. for all $\omega \in \Omega_2 \setminus \{\varepsilon\}$ and $d \in D_1$, "ω, d" belongs to Ω_2;

O_2 finite subset of Ω_2.

Formal description (2/2)

m-conf.	symbol	operations	final m-conf.
Q	S_1	O_2	Q

L_2 (finite) set $Q \times S_1 \times O_2 \times Q$ of well-formed lines;

T_2 (finite) set of (finite) sequences of elements of L_2 whose first two columns are unique.

Why use TC_2?

Why use TC_2?

- Lines are still fewer (division by $\sim \#Q$ w.r.t. TC_1 in the best case).

... However, Turing does not use TC_2! One can go further.

Example

- Right after describing TC_2, Turing gives an example... of another language!

m-conf.	symbol	operations	final *m*-conf.
b	None	P0, R	b
	0	R, R, P1	b
	1	R, R, P0	b

- Use of a curly brace (like indentation) to avoid repetition.
- TC_3 allows for multiple operations, i.e. containing an arbitrary number of elementary operations in any order (which retrospectively explains the merging of "writing" and "shifting" columns).

Formal description (1/2)

m-conf.	symbol	operations	final m-conf.
Q	S_1	O_3	Q

Ω_3 (infinite) set of operations of TC_3, defined by induction:

1. ε belongs to Ω_3;
2. for each $p \in P$, "p" belongs to Ω_3;
3. for each $d \in D_1$, "d" belongs to Ω_3;
4. for all $\omega \in \Omega_3 \setminus \{\varepsilon\}$ and $p \in P$, "ω, p" belongs to Ω_3;
5. for all $\omega \in \Omega_3 \setminus \{\varepsilon\}$ and $d \in D_1$, "ω, d" belongs to Ω_3;

O_3 finite subset of Ω_3.

Formal description (1/2)

m-conf.	symbol	operations	final m-conf.
Q	S_1	O_3	Q

L_3 (finite) set $Q \times S_1 \times O_3 \times Q$ of well-formed lines;

T_3 (finite) set of (finite) sequences of elements of L_3 whose first two columns are unique.

Why use TC_3?

Why use TC_3?

- Lines are still fewer (division by $\sim \#Q$ w.r.t. TC_1 even in suboptimal cases).

Example

- Turing finally gives "a slightly more difficult example" (section 3, p. 234): the machine II, which computes the sequence "001011011101111011111 0...".
- Implicitly, this machine is described in a new language (TC_4).

Example

m-c.	symbol	operations	final *m*-c.
b		Pə, R, Pə, R, P0, R, R, P0, L, L	o
o	1	R, Px, L, L, L	o
	0		q
q	Any (0 or 1)	R, R	q
	None	P1, L	p
p	x	E, R	q
	ə	R	f
	None	L, L	p
f	Any	R, R	f
	None	P0, L, L	o

Formal description (1/3)

m-conf.	symbol	operations	final *m*-conf.
Q	S_4	O_3	Q

S'_4 (finite) set of lists of no-empty symbols, defined by induction:

1. for each $s \in S \setminus \{None\}$, "$s$" belongs to S'_4;
2. for each $s' \in S'_4$ and for each $s \in S \setminus \{None\}$ which does not occur in s', "s' *or* s" belongs to S'_4;

Formal description (2/3)

m-conf.	symbol	operations	final *m*-conf.
Q	S_4	O_3	Q

S_4 (finite) set of symbol classes, defined by enumeration:

1. "None" belongs to S_4;
2. "Any" belongs to S_4 (= "Not None");
3. ε belongs to S_4 (= "None or Any");
4. for each $s \in S \setminus \{None\}$, "Not s" belongs to S_4;
5. for each $s' \in S'_4$, "s'" belongs to S_4.

Warning! None $\neq$ Any $\neq \varepsilon$

Formal description (3/3)

m-conf.	symbol	operations	final *m*-conf.
Q	S_4	O_3	Q

L_4 (finite) set $Q \times S_4 \times O_3 \times Q$ of well-formed lines;

T_4 (finite) set of (finite) sequences of elements of L_4 whose first two columns are unique (up to permutation of the elements of S'_4).

Why use TC_4?

q	None	P1, L	p
	0	L, L	q
	1	R, R	q
	2	R, R	q
	3	R, R	q
	...	...	...
	100	R, R	q

is replaced by

q	None	P1, L	p
	0	L, L	q
	Not 0	R, R	q

Why use TC_4?

Why use TC_4?

- Lines are still fewer (division by $\sim \#S$ even in suboptimal cases).

Introduction
TC_0: full and faithful description of the machine
TC_1: blind lines and double operations
TC_2: multiple shifting
TC_3: multiple operations
TC_4: classes of scanned symbols
Conclusion

Conclusion

In the 3rd section of Turing's article, one can distinguish between five different languages:

	Main feature	Example
TC_0	Faithful description of the machine	/
TC_1	Blind lines (S_1) + double operations (O_1)	1st machine I
TC_2	Multiple shifting (O_2)	/
TC_3	Multiple operations (O_3)	2nd machine I
TC_4	Classes of scanned symbols (S_4)	Machine II

Conclusion

	Gain
TC_0	
TC_1	Division by $\sim \#S$ in the best case
TC_2	Division by $\sim \#Q$ in the best case
TC_3	Division by $\sim \#Q$ in suboptimal cases
TC_4	Division by $\sim \#S$ in suboptimal cases

Introduction
TC_0: full and faithful description of the machine
TC_1: blind lines and double operations
TC_2: multiple shifting
TC_3: multiple operations
TC_4: classes of scanned symbols
Conclusion

Conclusion

- By eliminating repetitions, Turing progressively generalizes the machine's features:
  - it scans generalized symbols (S_4);
  - it performs generalized operations (O_1, O_2, O_3).
- However, *m*-configurations have not been generalized yet.
- This is what "abbreviated tables" will do.

Typology

1. Complete/Abbreviated tables ($TC_{0,\ 1,\ 2,\ 3,\ 4}$, $TA_{0,\ 1,\ 2}$)
   - explicative languages
   - intended for human readers and
   - allowing them to understand complex machines.
2. Standard description (TLa)
   - a computing language
   - intended for machines and
   - allowing them to handle machines as input/output.
3. Descriptive number (TLn)
   - an arithmetical language
   - intended for the mathematician and
   - allowing one to handle the cardinal and ordinal properties of computable numbers.

1 Introduction

2 Complete tables (TC)

3 Abbreviated tables (TA)

4 Linear alphabetic descriptions (TL_a)

5 Linear digital descriptions (TL_n)

6 Conclusion

Tables

- Two kinds of tables:
  1. complete tables: TC_0, TC_1, TC_2, TC_3, TC_4;
  2. abbreviated tables: TA_0, TA_1, TA_2.

Description

- In the 4th section, Turing describes a new family of machine descriptions: the "abbreviated tables" or "skeleton tables".

  There are certain types of process used by nearly all machines, and these, in some machines, are used in many connections. These processes include copying down sequences of symbols, comparing sequences, erasing all symbols of a given form, etc. Where such processes are concerned we can abbreviate the tables for the m-configurations considerably by the use of "skeleton tables".

Formal definition

m-conf.	symbol	operations	final *m*-conf.
F	S_4	O_3	F

F set of *m*-functions having

- as arguments: symbol names and *m*-configuration names;
- as value: *m*-functions.

m-functions

In skeleton tables there appear capital German letters and small Greek letters. These are of the nature of "variables". By replacing each capital German letter throughout by an m-configuration and each small Greek letter by a symbol, we obtain the table for an m-configuration.

- Abstraction over *m*-configurations and scanned symbols;
- application by substitution.

Abstraction over *m*-configurations

m-conf.	symbol	operations	final *m*-conf.
p		L	d
r		L	e
...		...	...
q		R	d
s		R	e
...		...	...

can be reduced to

m-conf.	symbol	operations	final *m*-conf.
l(E)		L	E
r(E)		R	E

Abstraction over scanned symbols

m-conf.	symbol	operations	final *m*-conf.
$c_1(E)$	0		pe(E,0)
$c_1(E)$	1		pe(E,1)
$c_1(E)$	x		pe(E,x)
$c_1(E)$	y		pe(E,y)
...	...		...

can be reduced to

m-conf.	symbol	operations	final *m*-conf.
$c_1(E)$	β		pe(E, β)

Skeleton tables as subroutines?

Thought of in this way, an analogy can be drawn between skeleton tables and the open subroutines or macro instructions found in later programming languages (Knuth and Pardo 1980, Copeland 2004). However, a richer historical understanding of Turing's motivation for the introduction and use of skeleton tables is gained by viewing it in the light of contemporary practice. As noted above, the strategy of defining complex functions in terms of simpler ones was commonly used by writers on recursive functions and effective computability. (Priestley 2011, p. 87)

Tables for humans

- Abbreviated tables are indeed intended for humans:

  The skeleton tables are to be regarded as nothing but abbreviations: they are not essential. So long as the reader understands how to obtain the complete tables from the skeleton tables, there is no need to give any exact definitions in this connection.

- Abbreviated tables are not programs: they are not supposed to be executed of compiled by a machine (even though Turing's thesis implies that a machine could do the job).

Library

- TA_1 is an extension of TA_0: a library of functions.
- Mnemonic names:

  f first or find;
  e erase
  pe print end.

- "Hungarian notation": names of variables and functions give information about their use:
  - global functions: f, q, l, r, re, e etc. (can be called from whatever function);
  - local functions: f_1, f_2 q_1, re_1 etc. (can be called only by another member of the same family: body of a function).

File system

- Those functions suppose the tape to be formated (a "file system", though without file segmentation):
  - beginning: ǝǝ;
  - end: two successive white spaces.

Comments

m-conf.	symbol	o.	final *m*-c.	
f(E,B,α)	ə	L	f_1(E,B,α)	From the *m*-configuration
	not ə	L	f(E,B,α)	*f(E,B,α)*, the machine finds
f_1(E,B,α)	α	R	E	the symbol of form α which
	not α	R	f_1(E,B,α)	is farthest to the left ("the
	None	R	f_2(E,B,α)	first α"), and the *m*-conf.
f_2(E,B,α)	α		E	then becomes E.
	not α	R	f_1(E,B,α)	If there is no α, then the
	None	R	B	*m*-configuration becomes B.

- Comments help to read (or not to read!) the code.

Good practice in programming

- B. Kernighan and Ph. J. Plauger, *The Elements of Programming Style*, 1978:
  - "use library functions";
  - "let the machine do the dirty job";
  - "replace repetitive expressions by calls to a common function";
  - "choose variable names that won't be confused";
  - "make your programs read from top to bottom";
  - "modularize; use subroutines";
  - "each module should do one thing well";
  - "make sure every module hides something";
  - "make sure comments and code agree"...

Which functions?

- Low-level functions;
- Intermediate functions;
- High-level functions.

Low-level functions: definition

Low-level functions functions which directly handle the tape (read, write, shift): massive "side effects".

Low-level functions: examples

m-conf.	symbol	operations	final *m*-conf.
q(E)	Any	R	q(E)
	None	R	q_1(E)
q_1(E)	Any	R	q(E)
	None		E
l(E)		L	E
r(E)		R	E

Low-level functions: use

- The four low-level functions perform shifting
  - absolute shifting: *f(E,B,α)* goes to the first occurrence of a symbol, *q(E)* to the end of tape;
  - relative shifting: *l* to left, *r* to right.

Intermediate functions: definition

Intermediate functions functions which directly handle the tape only in their private part (in the body of the function definition).

Intermediate functions: examples

m-conf.	symbol	operations	final *m*-conf.
$q(E,\alpha)$	$\emptyset$	$\emptyset$	$q(q_1(E,\alpha))$
$q_1(E,\alpha)$	α		E
	Not α	L	$q_1(E,\alpha)$
$e(E,B,\alpha)$	$\emptyset$	$\emptyset$	$f(e_1(E,B,\alpha),B,\alpha)$
$e_1(E,B,\alpha)$		E	E
$pe(E,\beta)$	$\emptyset$	$\emptyset$	$f(pe_1(E,\beta),E,$ ə$)$
$pe_1(E,\beta)$	Any	R,R	$pe_1(E,\beta)$
	None	Pβ	E

Intermediate functions: use

- Intermediate functions do searching and writing:
  - *q(E,α)* searches for the last occurrence of a symbol;
  - *e(E,B,α)* erases its first occurrence;
  - *pe(E,α)* writes it down at the end of tape;
  - *c(E,B,α)* copies a the end of tape the symbol preceding its first occurrence;
  - *re(E,B,α,β)* replaces its first occurrence by another symbol.

High-level functions: definition

High-level functions functions which do not handle directly the band.

High-level functions: examples

m-conf.	symbol	operations	final *m*-conf.
f'(E,B,α)	$\emptyset$	$\emptyset$	f(l(E),B,α)
f''(E,B,α)	$\emptyset$	$\emptyset$	f(r(E),B,α)
ce(E,B,α)	$\emptyset$	$\emptyset$	c(e(E,B,α),B,α)
ce(B,α)	$\emptyset$	$\emptyset$	ce(ce(B,α),B,α)
re(B,α,β)	$\emptyset$	$\emptyset$	re(re(B,α,β),B,α,β)
cr(E,B,α)	$\emptyset$	$\emptyset$	c(re(E,B,α,a),B,α)
cr(B,α)	$\emptyset$	$\emptyset$	cr(cr(B,α), re(B,a,α),α)
cpe(E_1,U,E,α,β)	$\emptyset$	$\emptyset$	cp(e(e(E_1,E,β),E,α),U,E,α

High-level functions: examples

- Examples:
  - *f'(E,B,α)* and *f''(E,B,α)*: go to the square preceding or following the first occurrence of a given symbol;
  - *ce(E,B,α)*, *ce(B,α)*, *cr(E,B,α)* and *cr(B,α)*: copy one or several symbols at the end of tape;
  - *re(B,α,β)* replaces all occurrences of a symbol by another;
  - *cpe(E_1,U,E,α,β)* and *cpe(U,E,α,β)* compare two strings of symbols.

High-level functions: use

- These are complex operations of
  - research
  - writing
  - moving
- ... and thus generalizations of elementary operations:
  - symbol scanning
  - symbol writing
  - tape shifting.
- Turing makes the central columns empty in order to generalize them.

A new kind of functions

- A line composed of high-level TA_1 functions does not any more perform "side effects" ($F \times S_4 \to O_3 \times F$): it associates one m-function to another ($F \to F$).
- High-level functions can be composed harmlessly.
- TA_1 becomes thus a functional language, just like λ-calculus!

m-conf.	symbol	operations	final m-conf.
ce(E,B,α)			c(e(E,B,α),B,α)

$$\simeq$$

$$\mathbf{ce} = \lambda E.\lambda B.\lambda \alpha.\mathbf{c}(\mathbf{e}(E, B, \alpha), B, \alpha)$$

Typed functions

- All functions are typed: some of them have the same name and differ only by their number or type of arguments: *e(E,B,*α*)*, *e(B,*α*)*, *e(E)*, *e(E,*α*,*β*)*.
- There are also several twin functions:

Operation	Unique usage	Iterated usage
symbol erasing	*e(E,B,*α*)*	*e(B,*α*)*
symbol moving	*ce(E,B,*α*)*	*ce(B,*α*)*
symbol replacement	*re(E,B,*α*,*β*)*	*re(B,*α*,*β*)*
symbol copy	*cr(E,B,*α*)*	*cr(B,*α*)*
symbol comparaison	*cpe(*E_1*,U,E,*α*,*β*)*	*cpe(U,E,*α*,*β*)*

Conclusion

TC_0-TC_4 descriptions based on the low-level behaviour (with side effects), with a decreasing number of lines;

TA_1-TA_2 high-level descriptions, based on a functional language which hides and generalizes the low-level ontology of the machines.

Typology

1. Complete/Abbreviated tables ($TC_{0,\ 1,\ 2,\ 3,\ 4}$, $TA_{0,\ 1,\ 2}$)
   - explicative languages
   - intended for human readers and
   - allowing them to understand complex machines.
2. Standard description (TLa)
   - a computing language
   - intended for machines and
   - allowing them to handle machines as input/output.
3. Descriptive number (TLn)
   - an arithmetical language
   - intended for the mathematician and
   - allowing one to handle the cardinal and ordinal properties of computable numbers.

1 Introduction

2 Complete tables (TC)

3 Abbreviated tables (TA)

4 Linear alphabetic descriptions (TL_a)

5 Linear digital descriptions (TL_n)

6 Conclusion

Two linear languages

- In section 5, Turing gives two new description languages, which are linear:

  TLa (alphabetical) standard descriptions (SD);
  TLn descriptive numbers (DN).

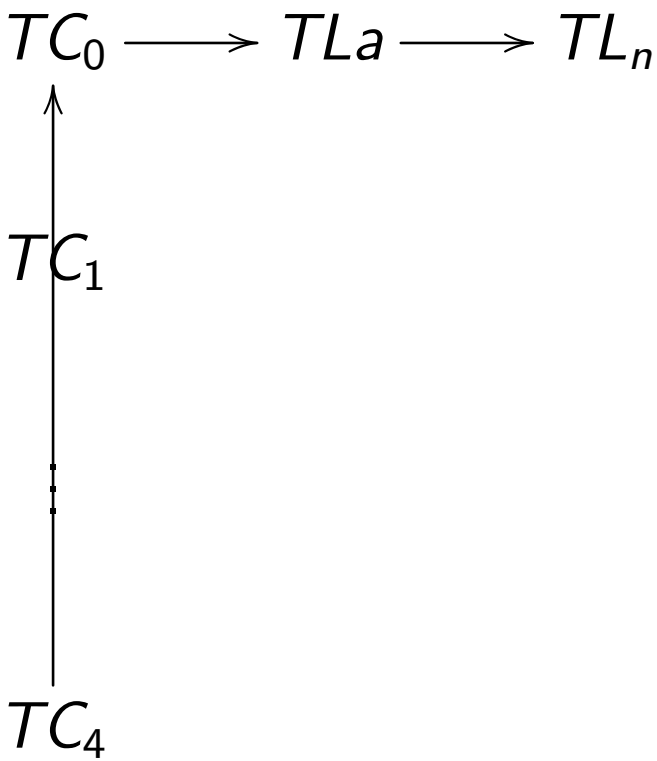
- These languages are bijective with each other but have different uses.

How to linearize a table

Turing goes back to a very low-level description of the machine:

> *It will be useful to put these tables into a kind of standard form. In the first place let us suppose that the table is given in the same form as the first table [...]. That is to say, that the entry in the operations column is always of one of the forms E :E,R:E,L:Pa: Pa, R: Pa, L:R:L: or no entry at all. The table can always be put into this form by introducing more m-conf..*

How to linearize a table



Renaming

- Renaming:
  - *m*-configurations: q_1, q_2, etc.
  - symbols: S_0 (empty space), S_1 ("0"), S_2 ("1"), etc.
  - PS_0 instead of E
- Saturation of columns:

m-conf.	symbol	operations	final *m*-conf.
q_i		R	q_m

becomes

m-conf.	symbol	operations	final *m*-conf.
q_i	S_0	PS_0, R	q_m
q_i	S_1	PS_1, R	q_m
q_i	S_2	PS_2, R	q_m

Example

m-conf.	symbol	operations	final *m*-conf.
b	None	P0,R	c
c	None	R	e
e	None	P1,R	f
f	None	R	b

becomes first ;$q_1S_0S_1Rq_2$;$q_2S_0S_0Rq_3$;$q_3S_0S_2Rq_4$;$q_4S_0S_0Rq_1$ and then ;DADDCR-DAA;DAADDRDAAA;DAAADDCCRDAAAA;DAAAADDRDA.

Use

- How does Turing concretely use this language?
  - only when a machine takes a description as input
- Examples:
  - U: the universal machine;
  - D: is the standard description that of a cyclic machine?
  - E: does the machine print a given symbol?
  - F: prints variants (up to substitution) of the standard description;
  - G: analyzes the results of F.
- TL_a is the only programming language in Turing's article: it is intended for machines.

1 Introduction

2 Complete tables (TC)

3 Abbreviated tables (TA)

4 Linear alphabetic descriptions (TL_a)

5 Linear digital descriptions (TL_n)

6 Conclusion

Bijection

- *TLn* and TL_a are bijective:

A	C	D	L	R	N	;
1	2	3	4	5	6	7

- Example: ";DADDCR-DAA;DAADDRDAAA;DAAADDCCRDAAAA;DAAAADDRDA" becomes

731332531173113353111731113322531111731111133531.

Use

However, *TLn* and TL_a have different uses.

- TL_a is used to perform operations on machine descriptions;
- *TLn* is only used to prove arithmetical (cardinal or ordinal) results.
  - e.g. enumerability of computable sequences:

To each computable sequence there corresponds at least one description number, while to no description number does there correspond more than one computable sequence. The computable sequences and numbers are therefore enumerable. (Section 5, p. 241)

A striking example

Let us suppose that there is such a process; that is to say, that we can invent a machine D which, when supplied with the S.D of any computing machine M will test this S.D and if M is circular will mark the S.D with the symbol "u" and if it is circle-free will mark it with "s". By combining the machines D and U we could construct a machine N to compute the sequence β'. The machine D may require a tape. We may suppose that it uses the E-squares beyond all symbols on F-squares, and that when it has reached its verdict all the rough work done by D is erased.

A striking example

The machine N has its motion divided into sections. In the first N-1 sections, among other things, the integers 1, 2,. . . , N-1 have been written down and tested by the machine D. A certain number, say R(N-1), of them have been found to be the D.N's of circle-free machines.

- Whey this switch from SD to DNs? Because Turing has to infer arithmetical properties from a program analysis.
- Different uses
  - TL_a for computation;
  - TL_n for arithmetical properties.

Typology

1. Complete/Abbreviated tables ($TC_{0,\ 1,\ 2,\ 3,\ 4}$, $TA_{0,\ 1,\ 2}$)
   - explicative languages
   - intended for human readers and
   - allowing them to understand complex machines.
2. Standard description (TLa)
   - a computing language
   - intended for machines and
   - allowing them to handle machines as input/output.
3. Descriptive number (TLn)
   - an arithmetical language
   - intended for the mathematician and
   - allowing one to handle the cardinal and ordinal properties of computable numbers.

1 Introduction

2 Complete tables (TC)

3 Abbreviated tables (TA)

4 Linear alphabetic descriptions (TL_a)

5 Linear digital descriptions (TL_n)

6 Conclusion

Ten languages for one machine

Turing uses several different languages to describe his machines:

1. a pyramid of *explicative languages*, from TC_0 to TC_4 and from TA_0 to TA_2, which are intended for human readers and progressively replace mechanical redundancies by intelligible features;
2. the *computing language TLa*, intended for the machine, which allows it to handle and product programs;
3. the *proving language TLn*, intended for the mathematician, which highlights and handles the cardinal and ordinal properties of computable numbers.

The universal machine

- The most striking example of this variety is the universal machine:
  - defined in TA_2, an extension of TA_1 (with memory initialization, boot function and an execution loop: a real operating system!);
  - designed to handle TL_a programs.
- Turing changes the description language depending on his goals.

Posterity

- Did Turing really create
  - machine language, assembly language, high-level programming languages, compilation
  - syntactic sugar, Hungarian notation
  - virtual machine
  - macros, procedures, pure functional languages, global and local functions, operator overloading
  - file systems
  - modularity
  - comments
  - specification
  - operating systems
  - program trace, etc.?

Posterity

- It would be too heavy to prove:
  - the absence of any precursor;
  - an intention;
  - a causal link with the successors.
- Historians doubt very much from such claims (Priestley, Mounier-Kuhn, Haigh, Daylight, De Mol, Bullynck...).
- Nevertheless, Turing did adopt the same strategy as the creators of computer programming languages: he constructed a hierarchy of languages corresponding to the variety of users and uses.

References 1/2

- Turing, A.M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem". Proceedings of the London Mathematical Society. 2 (published 1937). 42 (1): 230–65.
- Donald Knuth and Luis Trabb Pardo 1976/1980, "The Early Development of Programming Languages", Stanford U., Comp. Sci. Dept. (Aug. 1976); in N. Metropolis, J. Howlett, G.-C. Rota (eds.), *A History of Computing in the Twentieth Century*, San Diego (1980).
- B. Kernighan and Ph. J. Plauger (1978), *The Elements of Programming Style*.

References 2/2

- Jack Copeland (2004), "Computable numbers: a guide", *The Essential Turing*, pp. 5–57.
- Baptiste Mélès (2008), TMPL (Turing Machine Programming Language): `http://baptiste.meles.free.fr/?Le-langage-TMPL` (Perl implementation of TC_2).
- Mark Priestley (2011), *A Science of Operations*, ch. 4 and App. A.
- Liesbeth De Mol (2018), "Turing Machines", *The Stanford Encyclopedia of Philosophy*, `https://plato.stanford.edu/entries/turing-machine/`.