Solutions to Selected Exercises

Chapter 2

2.1. (a) *Regular.* Every production has a lone non-terminal on its left-hand side, and the right-hand sides consist of either a single terminal, or a single terminal followed by a single non-terminal.

(b) *Context free.* Every production has a lone non-terminal on its left-hand side. The right-hand sides consist of arbitrary mixtures of terminals and/or non-terminals one of which (*aAbb*) does not conform to the pattern for regular right-hand sides.

2.2. (a) $\{a^{2i}b^{3j}: i, j \geqslant 1\}$ i.e. *a*s followed by *b*s, any number of *a*s divisible by two, any number of *b*s divisible by three

(b) $\{a^i a^j b^{2j}: i \geqslant 0, j \geqslant 1\}$ i.e. zero or more *a*s, followed by one or more *a*s followed by twice as many *b*s

(c) { } i.e. the grammar generates no strings at all, as no derivations beginning with *S* produce a terminal string

(d) $\{\varepsilon\}$ i.e. the only string generated is the empty string.

2.3. *xyz*, where $x \in (N \cup T)^*$, $y \in N$, and $z \in (N \cup T)^*$

The above translates into: "a possibly empty string of terminals and/or non-terminals, followed by a single non-terminal, followed by another possibly empty string of terminals and/or non-terminals".

2.5. For an alphabet A, A^* is the set of all strings that can be taken from A including the empty string, ε. A regular grammar to generate, say $\{a, b\}^*$ is

$$S \rightarrow \varepsilon \mid aS \mid bS.$$

ε is derived directly from S. Alternatively, we can derive *a* or *b* followed by *a* or *b* or ε (this last case terminates the derivation), the *a* or *b* from the last

stage being followed by a or b or ε (last case again terminates the derivation), and so on ...
Generally, for any alphabet, $\{a_1, a_2, \ldots, a_n\}$ the grammar

$$S \to \varepsilon \mid a_1S \mid a_2S \mid \ldots \mid a_nS$$

is regular and can be similarly argued to generate $\{a_1, a_2, \ldots, a_n\}^*$.

2.7. (b) The following fragment of the BNF definition for Pascal, taken from Jensen and Wirth (1975), actually defines all Pascal expressions, not only Boolean expressions.

<expression> ::= <simple expression> | <simple expression>
<relational operator> <simple expression>
<simple expression> ::= <term> | <sign> <term> |
<simple expression> <adding operator> <term>
<adding operator> ::= + | − | or
<term> ::= <factor> | <term> <multiplying operator> <factor>
<multiplying operator> ::= * | / | div | mod | and
<factor> ::= <variable> | <unsigned constant> | (<expression>) |
<function designator> | <set> | not <factor>
<unsigned constant> ::= <unsigned number> | <string> |
<constant identifier> | nil
<function designator> ::= <function identifier> |
<function identifier> (<actual parameter>
{, <actual parameter>})
<function identifier> ::= <identifier>
<variable> ::= <identifier>
<set> ::= [<element list>]
<element list> ::= <element> {, <element> } | <empty>
<empty> ::=

Note the use of the empty string to enable an empty <set> to be specified (see Chapter 5).

2.10. Given a finite set of strings, each string x, from the set can be generated by regular grammar productions as follows:

$$x = x_1x_2 \ldots x_n, \quad n \geqslant 1$$
$$S \to x_1X_1$$
$$X_1 \to x_2X_2$$
$$\vdots$$
$$X_{n-1} \to x_n.$$

For each string, x, we make sure the non-terminals X_i, are unique (to avoid any derivations getting "crossed").

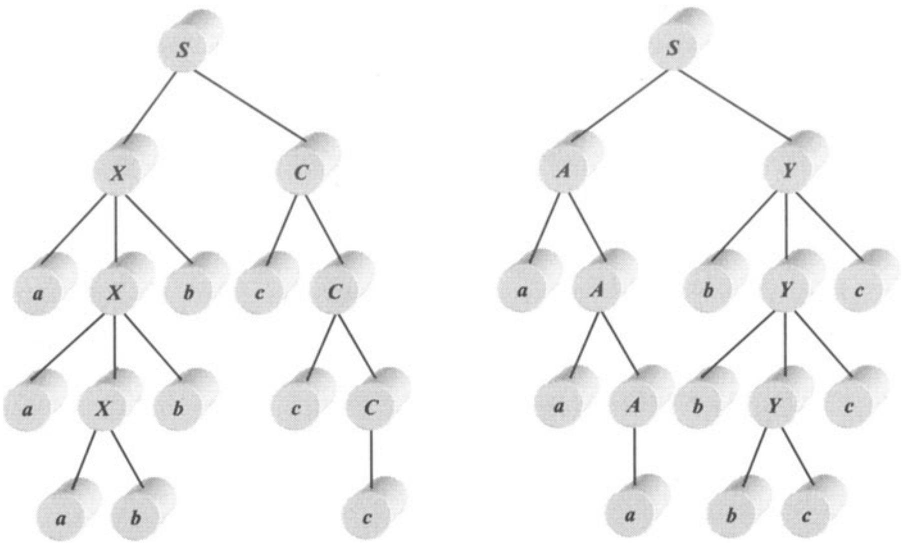

Figure S.1. Two derivation trees for the same sentence ($a^3b^3c^3$)

This applies to any finite set of strings, so any finite set of strings is a regular language.

Chapter 3

3.1. (b) $\{a^ib^jc^k: i, j, k \geqslant 1, i = j \text{ or } j = k\}$

i.e. strings of *a*s followed by *b*s followed by *c*s, where the number of *a*s equals the number of *b*s, or the number of *b*s equals the number of *c*s, or both

(c) Grammar *G* can be used to draw two different derivation trees for the sentence $a^3b^3c^3$, as shown in Figure S.1.

The grammar is thus ambiguous.

3.2. (b) As for the Pascal example, the semantic implications should be discussed in terms of demonstrating that the same statement yields different results according to which derivation tree is chosen to represent its structure.

Chapter 4

4.1. (a) The FSR obtained directly from the productions of the grammar is shown in Figure S.2.

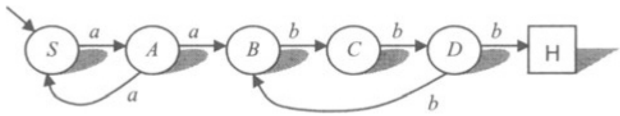

Figure S.2. A non-deterministic finite state recogniser

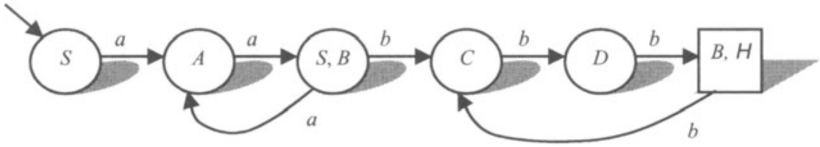

Figure S.3. A deterministic version of the FSR in Figure S.2

The FSR in Figure S.2 is non-deterministic, since it contains states (for example, state A) with more than one identically labelled outgoing arcs going to different destinations.

The deterministic version, derived using the subset method (null state removed) is in Figure S.3.

4.3. One possibility is to represent the FSR as a two-dimensional table (array), indexed according to (state, symbol) pairs. Table S.1 represents the FSR of Exercise 1(a) (Figure S.2).

In Table S.1, element (A, a), for example, represents the set of states ($\{S, B\}$) that can be directly reached from state A given the terminal a. Such a representation would be easy to create from the productions of a grammar that could be entered by the user, for example. The representation is also highly useful for creating the deterministic version. This version is also made more suitable if the language permits dynamic arrays (Pascal does not, but ADA, C, and Java are languages that do). The program also needs to keep details of which states are halt and start states.

An alternative scheme represents the FSR as a list of triples, each triple representing one arc in the machine. In languages such as Pascal or ADA, this

Table S.1. A tabular representation of the finite state recogniser in Figure S.2

	Terminal symbols	
States	a	b
S	A	–
A	S, B	–
B	–	C
C	–	D
D	–	B, H
H	–	–

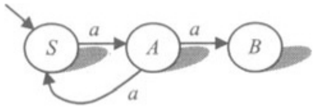

Figure S.4. Part of the finite state recogniser from Figure S.2

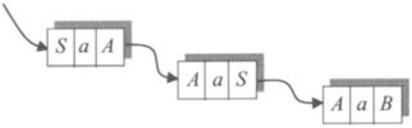

Figure S.5. The FSR fragment from Figure S.4 represented as a linked list of (state, arc symbol, next state) triples

scheme can be implemented in linked list form. For example, consider the part of the FSR from Exercise 4.1 (Figure S.2) shown in Figure S.4.

This can be represented as depicted in Figure S.5.

This representation is particularly useful when applying the reverse operation in the minimisation algorithm (the program simply exchanges the first and third elements in each triple). It is also a suitable representation for languages such as LISP or PROLOG, where the list of triples becomes a list of three element lists.

Since FSRs can be of arbitrary size, a true solution to the problem of defining an appropriate data structure would require dynamic data structures, even down to allowing an unlimited source of names. Although this is probably taking things to extremes, you should be aware when you are making such restrictions, and what their implications are.

Chapter 5

5.2. First of all, consider that a DPDR is not necessarily restricted to having one halt state. You design a machine, *M*, that enters a halt state after reading the first *a*, then remains in that state while reading any more *a*s (pushing them on to the stack, to compare with the *b*s, if there are any). If there are no *b*s, *M* simply stops in that halt state, otherwise on reading the first *b* (and popping off an *a*) it makes a transition to another state where it can read only *b*s. The rest of *M* is an exact copy of M_3^d, of Chapter 5. If there were no *b*s, any *a*s on the stack must remain there, even though *M* is accepting the string. Why can we not ensure that in this situation, *M* clears the *a*s from its stack, but is still deterministic?

5.5. The reasons are similar to why arbitrary palindromic languages are not deterministic. When the machine reads the *a*s and *b*s part of the string, it has no way of telling if the string it is reading is of the "number of *a*s = number of *b*s", or the "number of *b*s = number of *c*s" type. It thus has to assume that the input string is of the former type, and backtrack to abandon this assumption if the string is not.

5.6. One possibility is to represent the PDR in a similar way to the list representation of the FSR described above (sample answer to Exercise 4.3, Chapter 4). In the case of the PDR, the "current state, input symbol, next state" triples would become "quintuples" of the form:

current state, input sym, pop sym, push string, new state

The program could read in a description of a PDR as a list (a file perhaps) of such "rules", along with details of which states were start and halt states.

The program would need an appropriate dynamic data structure (for example, linked list) to represent a stack. It may therefore be useful to design the stack and its operations first, as a separate exercise.

Having stored the rules, the program would then execute the algorithm in Table S.2.

As the PDR is deterministic, the program can assume that only one quintuple will be applicable at any stage, and can also halt its processing of invalid strings as soon as an applicable quintuple cannot be found. The non-deterministic machine is much more complex to model: I leave it to you to consider the details.

Table S.2. An algorithm to simulate the behaviour of a deterministic push down recogniser. The PDR is represented as quintuples

```
can-go := true
C := the start state of the PDR

while not(end-of-input) and can-go
  if there is a quintuple, Q, such that
    Q's current state = C, and
    Q's pop sym = the current symbol "on top" of the stack, and
    Q's read sym = the next symbol in the input
  then
    remove the top symbol from the stack
    set up ready to read the next symbol in the input
    push Q's push string onto the stack
    set C to be Q's next state
  else
    can-go := false
  endif
endwhile

if end-of-input and C is a halt state then
  return("yes")
else
  return("no")
endif
```

Chapter 6

6.3. **Any FSR that has a loop on some path linking its start and halt states *in which there is one or more arcs not labelled with* ε recognises an infinite language.**

6.4. **In both cases, it is clear that the *v* part of the *uvw* form can consist only of *a*s or *b*s or *c*s. If this were not the case, we would end up with symbols out of their respective correct order. Then one simply argues that when the *v* is repeated the required numeric relationship between *a*s, *b*s, and *c*s is not maintained.**

6.5. **The language specified in this case is the set of all strings consisting of two copies of any string of *a*s and/or *b*s. To prove that it is not a CFL, it is useful to use the fact that we know we can find a *uvwxy* form for which $|vwx| \leqslant 2^n$, and n is the number of non-terminals in a Chomsky Normal Form grammar to generate our language. Let $k = 2^n$. There are many sentences in our language of the form $a^n b^n a^n b^n$, where $n > k$. Consider the *vwx* form as described immediately above. I leave it to you to complete the proof.**

Chapter 7

7.1. **This can be done by adding an arc labelled *x*/*x* (*N*) for each symbol, *x* in the alphabet of the particular machine (including the blank) from each of the halt states of the machine to a new halt state. The original halt states are then designated as non-halt states. The new machine reaches its single halt state leaving the tape/head configuration exactly as did the original machine.**

Chapter 8

8.1. **(The "or" FST). Assuming that the two input binary strings are of the same length, and are interleaved on the tape, a bitwise *or* FST is shown in Figure S.6.**

8.2. **An appropriate FST is depicted in Figure S.7.**

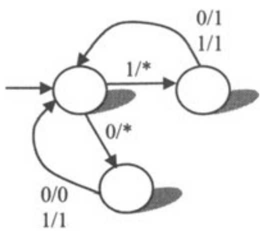

Figure S.6. A bitwise "or" finite state transducer. The two binary numbers are the same length, and interleaved when presented to the machine. The machine outputs "*" on the first digit of each pair

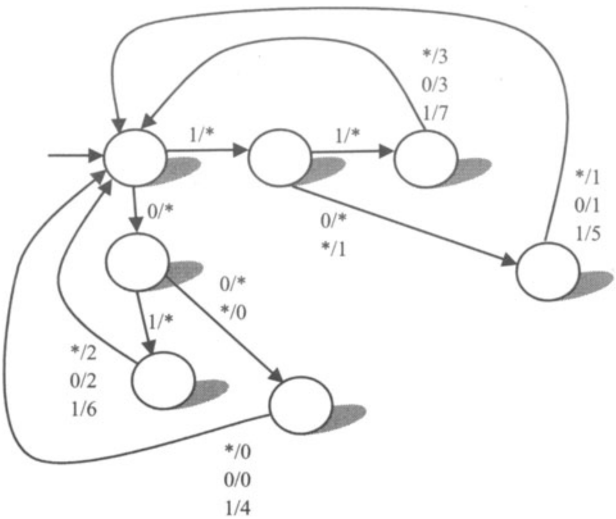

Figure S.7. A finite state transducer that converts a binary number into an octal number. The input number is presented to the machine in reverse, and terminated with "*". The answer is also output in reverse

Chapter 9

9.4. Functions are discussed in more detail in Chapter 11. A specification of a function describes what is computed and does not go into detail about how the computation is done. In this case, then, the TM computes the function:

$$f(y) = y \; div \; 2, y \geqslant 1.$$

9.5. (a) tape on entry to loop: $d1^{x+1}e$

tape on exit: $df^{x+1}e1^{x+1}$ i.e. the machine copies the $x + 1$ 1s between d and e to the right of e, replacing the original 1s by fs.

(b) $f(x) = 2x + 3$.

Chapter 10

10.3. The sextuple (1, *a*, *A*, *R*, 2, 2) of the three tape machine *M* might be represented as shown in Figure S.8.

Chapter 11

11.1. Assume the TM, *P*, solves the printing problem by halting with output 1 or 0 according to whether *M* would, or would not, write the symbol *s*.

P would need to be able to solve the halting problem, or in cases where *M* was not going to halt *P* would not be able to write a 0, for "no".

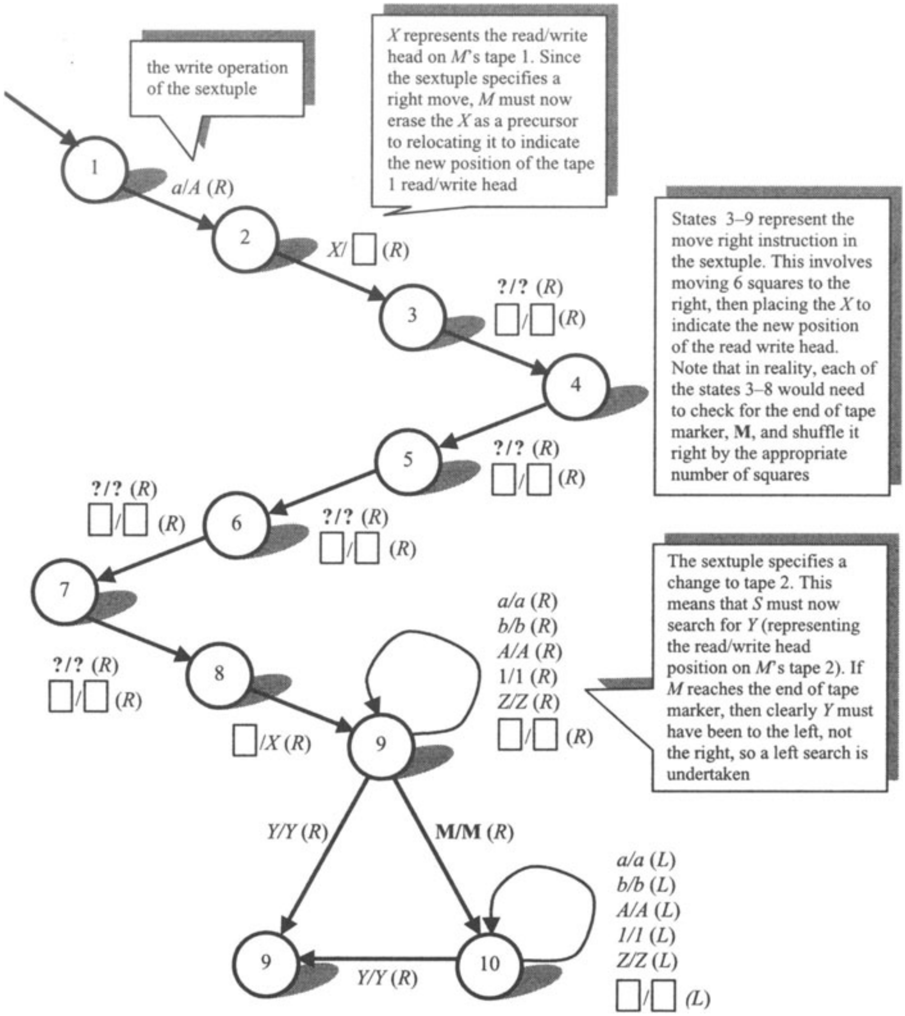

Figure S.8. A sketch of how a single tape TM, S, could model the sextuple $(1, a, A, R, 2, 2)$ of a 3-tape machine, M, from Chapter 10. For how the 3 tapes are coded onto S's single tape, see Figures 10.21 and 10.22. This sequence of states applies when S is modelling state 1 of M and S's read write head is on the symbol representing the current symbol of M's tape 1

Chapter 12

12.1. There are two worst case scenarios. One is that the element we are looking for is in position $(n, 1)$, i.e. the leftmost element of the last row. The other is that the element is not in the array at all, but is greater than every element at the end of a row except the one at the end of the last row, and smaller than every element on the last row. In both of these cases we inspect every element at the end of a row (there are n of these), then every element on the

last row (there are $n-1$ of these, as we already inspected the last one). Thus, we inspect $2n-1$ elements. This represents time $O(n)$.

There are also two average case scenarios. One is that the element is found midway along the middle row. The other is that the element is not in the array at all, but is greater than every element at the end of a row above the middle row, smaller than every element in the second half of the middle row and greater than the element to the left of the middle element in the middle row. In this case, we inspect half of the elements at the end of the rows (there are $n/2$ of these), and we then inspect half of the elements on the middle row (there are $n/2 - 1$) of these, since we already inspected the element at the end of the middle row. We thus make $n/2 + n/2 - 1 = n - 1$ comparisons. This, once again, is $O(n)$.

12.3. (c) A further form of useless state, apart from those from which the halt state cannot be reached, is one that cannot be reached from the start state. To find these, we simply examine the row in the connectivity matrix for the start state, S, of the machine. Any entry on that row (apart from position S, S) that is not a 1 indicates a state that cannot be reached from S, and is thus useless. If there are n states in the machine, this operation requires time $O(n)$.

With respect to part (b) of the question, the *column* for a state indicates the states from which that state can be reached. Entries that are not 1 in the column for the halt state(s) indicate states from which the halt state cannot be reached (except, of course for entry H, H where H is the halt state in question). This operation is $O(m \times n)$ where n is the total number of states, and m is the number of halt states. The running time is thus never worse that $O(n^2)$ which would be the case if all states were halt states. For machines with a single halt state it is, of course $O(n)$.

12.4. Table S.3 shows the result of applying the subset algorithm (Table 4.6) to the finite state recogniser of Figure 12.10. For an example see the finite state recogniser in Figure S.2, which is represented in tabular form in Table S.1.

Table S.3. The deterministic version of the finite state recogniser from Figure 12.10, as produced by the subset algorithm of Chapter 4 (Table 4.6). There are eight states, which is the maximum number of states that can be created by the algorithm from a 3-state machine

	Terminal symbols					
States	*a*	*b*	*c*	*d*	*e*	*f*
1 (start state)	1_2	1_2_3	*N*	1_3	3	2
2	*N*	*N*	2_3	*N*	*N*	*N*
3 (halt state)	*N*	*N*	*N*	*N*	*N*	*N*
1_2	1_2	1_2_3	2_3	1_3	3	2
1_3 (halt state)	1_2	1_2_3	*N*	1_3	3	2
2_3 (halt state)	*N*	*N*	2_3	*N*	*N*	*N*
1_2_3 (halt state)	1_2	1_2_3	2_3	1_3	3	2
N (null state)	*N*	*N*	*N*	*N*	*N*	*N*

Here, the tabular form is used in preference to a diagram, as the machine has a rather complex structure when rendered pictorially.

Chapter 13

13.1. The *NOR* gate representation of the *and* operator is depicted in Figure S.9 (cf. de Morgan's law for the intersection of two sets, in Figure 6.7). The truth table in Table S.4 verifies the representation. Note the similarity between this representation and the *NAND* gate representation of the *or* operator given in Figure 13.5.

13.2. (a) Figure S.10 shows the engineer's circuit built up entirely from *NAND* gates. Note that we have taken advantage of one of de Morgan's laws (Table 13.12), to express $C + \sim D$ as $\sim(\sim C \,.\, D)$, i.e. $(\sim C$ *NAND* $D)$. This involves an intermediate stage of expressing $C + \sim D$ as $\sim\sim(C + \sim D)$ – rule 1 of Table 13.12.

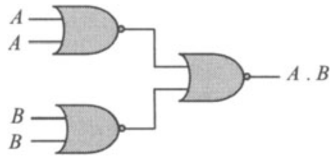

Figure S.9. The *NOR* gate representation of the Boolean and (.) operator

Table S.4. Verification by truth table that the *NOR* gate representation, $\sim(\sim(A + A) + \sim(B + B))$ is equivalent to $A \,.\, B$

A	B	(*P*) $A + A$	(*R*) $\sim P$	(*Q*) $B + B$	(*S*) $\sim Q$	(*T*) $R + S$	(*Result* 1) $\sim T$	(*Result* 2) $A \,.\, B$
0	0	0	1	0	1	1	0	0
0	1	0	1	1	0	1	0	0
1	0	1	0	0	1	1	0	0
1	1	1	0	1	0	0	1	1

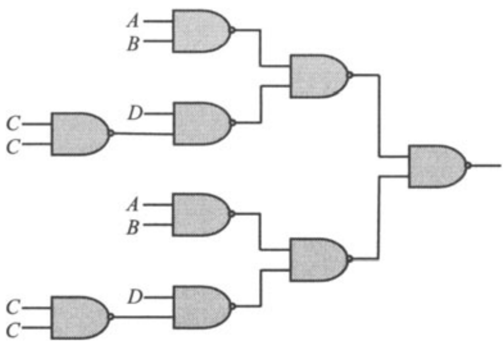

Figure S.10. The engineer's circuit of Chapter 13, i.e. $\sim(A \,.\, B) \,.\, (C + \sim D)$, using only *NAND* gates

Table S.5. Verification by truth table that $p \wedge p$ is equivalent to p

p	$p \wedge p$
0	0
1	1

13.5. (c) The truth table showing the equivalence of ($p \wedge p$) with p is given in Table S.5.

The significance of this is that whenever we have a proposition of this form, for example ($s \rightarrow t$) $\wedge$ ($s \rightarrow t$), we can simply remove one of the equal parts, in this case giving ($s \rightarrow t$).

Chapter 14

14.5. The representation of the problem statement, and the reasoning required to solve the problem, are given in Table S.6.

Table S.6. Classical reasoning in FOPL

FOPL	Comments
$A1.\ (\forall x)\ (politician(x) \rightarrow (liar(x) \vee cheat(x)))$	Axioms
$A2.\ (\forall y)((married(y) \wedge cheat(y)) \rightarrow affair(y))$	
$A3.\ \sim affair(Alg)$	
$A4.\ \sim liar(Alg)$	
P: $married(Alg) \rightarrow \sim politician(Alg)$	Statement to be proved
$politician(Alg) \rightarrow (liar(Alg) \vee cheat(Alg))$	From $A1$
$\sim politician(Alg) \vee (liar(Alg) \vee cheat(Alg))$	$(p \rightarrow q) \leftrightarrow (\sim p \vee q)$
$\sim politician(Alg) \vee cheat(Alg)$	(X) With $A4$: $p \vee 0 \leftrightarrow p$ (see Exercise 5(e), Chapter 13)
$politician(Alg) \rightarrow cheat(Alg)$	$(p \rightarrow q) \leftrightarrow (\sim p \vee q)$
$(married(Alg) \wedge cheat(Alg)) \rightarrow affair(Alg)$	From $A2$
$\sim(married(Alg) \wedge cheat(Alg))$	With $A3$: *modus tollens*
$\sim married(Alg) \vee \sim cheat(Alg)$	de Morgan's law
$cheat(Alg) \rightarrow \sim married(Alg)$	(Y) $(p \rightarrow q) \leftrightarrow (\sim p \vee q)$
$(politician(Alg) \rightarrow cheat(Alg)) \wedge (cheat(Alg) \rightarrow \sim married(Alg))$	$X \wedge Y$
$politician(Alg) \rightarrow \sim married(Alg)$	Transitivity of implication (see Exercise 5(a), Chapter 13)
$\sim politician(Alg) \vee \sim married(Alg)$	$(p \rightarrow q) \leftrightarrow (\sim p \vee q)$
$\sim married(Alg) \vee \sim politician(Alg)$	
$married(Alg) \rightarrow \sim politician(Alg)$	P is proved

Chapter 15

15.1. The representation of the problem statement, and the resolutions required to solve the problem, are given in Table S.7.

Table S.7. Proof by resolution. This is the same problem as Exercise 14.5 of Chapter 14

Proof	Comments
$A1.\ (\forall x)(politician(x) \rightarrow (liar(x) \vee cheat(x)))$	Axioms
$A2.\ (\forall y)(married(y) \wedge cheat(y)) \rightarrow affair(y))$	
$A3.\ {\sim}affair(Alg)$	
$A4.\ {\sim}liar(Alg)$	
$P\!:\ married(Alg) \rightarrow\ {\sim}politician(Alg)$	Statement to be proved
$C1.\ {\sim}politician(x) \vee liar(x) \vee cheat(x)$	From $A1$
$C2.\ {\sim}married(y) \vee {\sim}cheat(y) \vee affair(y)$	From $A2$
$C3.\ {\sim}affair(Alg)$	$A3$
$C4.\ {\sim}liar(Alg)$	$A4$
$C5.\ married(Alg)$	From ${\sim}P$ (negated statement to be proved)
$C6.\ politician(Alg)$	
$C7.\ {\sim}politician(x) \vee liar(x) \vee {\sim}married(x) \vee affair\ (x)$	$C1$ and $C2$ resolved
$C8.\ liar(Alg) \vee {\sim}married(Alg) \vee affair(Alg)$	$C6$ and $C7$ resolved
$C9.\ liar(Alg) \vee affair(Alg)$	$C8$ and $C5$ resolved
$C10.\ affair(Alg)$	$C9$ and $C4$ resolved
$C11.$ <empty>	$C10$ and $C3$ resolved. P is proved

15.2. The representation of the problem statement, and the resolutions required to solve the problem, are given in Table S.8.

Table S.8. Using the transitivity of the *greater than* relation to prove that $26 > 1$ by resolution

Proof	Comments
$A1.\ (\forall x, y, z)(p(x, y) \wedge p(y, z)) \rightarrow p(x, z))$	Axioms
$A2.\ p(24, 3)$	$24 > 3$
$A3.\ p(26, 24)$	$26 > 24$
$A4.\ p(3, 1)$	$3 > 1$
$P.\ p(26, 1)$	Statement to be proved
$C1.\ {\sim}p(x, y) \vee {\sim}p(y, z) \vee p(x, z)$	From $A1$
$C2.\ p(24, 3)$	$A2$
$C3.\ p(26, 24)$	$A3$
$C4.\ p(3, 1)$	$A4$
$C5.\ {\sim}p(26, 1)$	${\sim}P$ (negated statement to be proved)
$C6.\ {\sim}p(24, z) \vee p(26, z)$	$C1$ and $C3$ resolved
$C7.\ p(26, 3)$	$C6$ and $C2$ resolved
$C8.\ {\sim}p(3, z) \vee p(26, z)$	$C7$ and $C1$ resolved
$C9.\ p(26, 1)$	$C8$ and $C4$ resolved
$C10.$ <empty>	$C9$ and $C5$ resolved. P is proved

Further Reading

The following are some suggested titles for further reading. Notes accompany most of the items. Some of the titles refer to articles that describe practical applications of concepts from this book.

The numbers in parentheses in the notes refer to chapters in this book.

Church A. (1936) An Unsolvable Problem of Elementary Number Theory. American Journal of Mathematics, 58, 345–363.

Church and Turing were contemporaneously addressing the same problems by different, but equivalent means. Hence, in books such as Harel's we find references to the "Church–Turing thesis", rather than "Turing's thesis". (9–11).

Cohen D. I. A. (1996) Introduction to Computer Theory. John Wiley, New York. 2nd edition.

Covers some additional material such as regular expressions, Moore and Mealy machines (in this book our FSTs are Mealy machines) (8). Discusses relationship between multi-stack PDRs (5) and TMs.

Floyd R.W. and Beigel R. (1994) The Language of Machines: An Introduction to Computability and Formal Languages. W.H. Freeman, New York.

Includes a discussion of regular expressions (4), as used in the UNIX™ utility "egrep". Good example of formal treatment of minimisation of FSRs (using equivalence classes)(4).

Harel D. (1992) Algorithmics: The Spirit of Computing. Addison-Wesley, Reading, MA. 2nd edition.

Study of algorithms and their properties, such as complexity, big O running time (12) and decidability (11). Discusses application of finite state machines to modelling simple systems (8). Focuses on 'counter programs': simple programs in a hypothetical programming language.

Harrison M.A. (1978) Introduction to Formal Language Theory. Addison-Wesley, Reading, MA.

Many formal proofs and theorems. Contains much on closure properties of languages (6).

Hopcroft J.E. and Ullman J.D. (1979) Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, MA.

Discusses linear bounded TMs for context sensitive languages (11).

Jensen K. and Wirth N. (1975) Pascal User Manual and Report. Springer-Verlag, New York.

Contains the BNF and Syntax chart descriptions of the Pascal syntax (2). Also contains notes referring to the ambiguity in the "if" statement (3).

Kain R.Y. (1972) Automata Theory: Machines and Languages. McGraw-Hill, New York.

Formal treatment. Develops Turing machines before going on to the other abstract machines. Discusses non-standard PDRs (5) applied to context sensitive languages.

Minsky M.L. (1967) Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ.

A classic text, devoted to an investigation into effective procedures (11). Very detailed on most aspects of computer science. Of particular relevance is description of Shannon's 2-state TM result (12), and reference to unsolvable problems (11). The proof we use in this book to show that FSTs cannot perform arbitrary multiplication (8) is based on Minsky's.

Murdocca M. (2000) Principles of Computer Architecture. Addison Wesley, Reading, MA.

Computer architecture books usually provide useful material on Boolean logic and its application in digital logic circuits (13). This book also has sections on reduction of logical circuits.

Kelley D. (1998) Automata and Formal Languages: An Introduction. Prentice Hall, London.

Covers most of the introductory material on regular (4) and context free (5) languages, also has chapters on Turing machine language processing (7), decidability (11) and computational complexity (12).

Post E. (1936) Finite Combinatory Processes – Formulation 1. Journal of Symbolic Logic, 1, 103–105.
Post formulated a simple abstract string manipulation machine at the same time as did Turing (9). Cohen (see above) devotes a chapter to these "Post" machines.

Rayward-Smith V.J. (1983) A First Course in Formal Language Theory. Blackwell, Oxford, UK.
The notation and terminology for formal languages we use in this book is based on Rayward-Smith. Very formal treatment of regular languages (plus regular expressions), FSRs (4), and context free languages and PDRs (5). Includes Greibach normal form (as does Floyd and Beigel) an alternative CFG manipulation process to Chomsky Normal Form (5). Much material on top-down and bottom-up parsing (3), LL and LR grammars (5), but treatment very formal.

Rich E. and Knight K. (1991) Artificial Intelligence. McGraw-Hill, New York.
Artificial intelligence makes much use of representations such as grammars and abstract machines. In particular, machines called recursive transition networks and augmented transition networks (equivalent to TMs) are used in natural language processing. AI books are usually good for learning about first order predicate logic and resolution (14–15), since AI practitioners are interested in using FOPL to solve real world reasoning problems.

Tanenbaum A.S. (1998) Computer Networks. Prentice-Hall, London. 3rd edition.
Discusses FSTs (8) for modelling protocol machines (sender or receiver systems in computer networks).

Turing A. (1936) On Computable Numbers with an Application to the Entscheidungs problem. Proceedings of the London Mathematical Society, 42, 230–265.
The paper in which Turing introduces his abstract machine, in terms of computable numbers rather than computable functions. Also includes his notion of a universal machine (10). A paper of remarkable contemporary applicability, considering that Turing was considering the human as computer, and not machines.

Winston P.H. (1992) Artificial Intelligence. Addison-Wesley, Reading, MA (see Rich), 3rd edition.

Wood D. (1987) Theory of Computation. John Wiley, Chichester, UK.
Describes several extensions to PDRs (5). Introduction to proof methods, including the pigeonhole principle (also mentioned by Harel) on which both the repeat state theorem (6, 8) and the *uvwxy* theorem (6) are based.

Index

abstract machine 1–3, 133
computational model 117, 130, 136, 204
efficiency 250, 256
model of time 249, 251, 256
clock 166, 251–252
relationship to languages 42, 232
acceptable language 151–158, 228, 242–245
ADA [programming language] 330
Algol 44
Algol60 42, 45
Algol68 44
algorithm
complexity 4
termination condition 79, 240
theoretical significance 270, 289
see also running time of algorithm
alphabet 11–12, 90, 108, 165–167
as a set 11
finiteness of 11
ambiguity 37, 42–47, 295
ambiguous grammar 50
definition of 45
ambiguous language
definition of 42
general lack of solution 45
implications
compilation 43–45
logical expressions 46
programming language definition 44–45
programs 45
semantic 43–44, 46, 329
inherently ambiguous language 46
natural language 42, 45
Krushchev 45
Pascal 42–44
solution 44–45
unambiguous grammar 45
unambiguous language 44
unambiguous logical expressions 46
array 12–13, 256–257, 263, 271–272
dynamic 13, 330
indexes 330
one dimensional 256
sorted 256–257
three dimensional 255
two dimensional 199, 255, 261
see also search algorithm, sort algorithm *and* Turing machine
asymmetric relation 306
atomic symbol 11
automaton 1, 245
axiom 316

Backtracking 61, 102, 148, 251, 305, 323, 332
computational problems of 61
Backus-Naur form 20–22, 36, 43, 328
big O analysis 249, 254–258, 260–269, 270–272, 335–336
dominant expression 255, 267
see also running time of algorithm
binary number system, universality of 172, 196–197, 232, 250–251
binary search 258–260
analogous to telephone directory 257
vs. linear search 260
worst case 260
see also big O analysis, linear one dimensional array search, running time of algorithm
binary tree 89
blocks world problem 299–300, 306
BNF, *see* Backus-Naur form
Boole, George 275
Boolean logic 1, 4, 47, 178, 275–282, 284
circuit 178, 276
expression (statement) 36, 47, 275–276, 279–281, 293, 328
operator 4, 46–47, 178, 275–276, 278–281
and 276, 337
not 276
or 276, 337
problem solving 276–278, 287
role in computing 278–281
digital circuitry 278
information retrieval 280–281

Boolean logic (*continued*)
programming languages 279
variable 275
bubble (exchange) sort 263, 271
best case 263–264, 271
worst case 263–264

C [programming language] 330
CFG, *see* context free grammar
CFL, *see* context free language
Chomsky hierarchy 2, 11, 26, 30–32, 35, 49, 81, 103, 107, 130–131, 145, 155, 158, 179, 213, 242, 245–247
classification of grammars 30–32, 33, 35
Chomsky Noam 2, 26
see also Chomsky hierarchy *and* Chomsky Normal Form
Chomsky normal form for CFGs 85, 89, 105, 122, 127–128, 130
equivalence with CFGs 85, 89
Church Alonzo 204
clausal database, *see* conjunctive normal form database
clever two dimensional array search 262–263, 271
average case 271, 336
worst case 271, 335–336
see also big O analysis, linear two dimensional array search, running time of algorithm
closure (of language) 107–118, 122
see also context free language *and* regular language
CNF, *see* Chomsky normal form
compiler 1–2, 18–19, 37, 40, 43–44, 45, 68, 82, 145, 197, 207, 234
see also ambiguity, decision program *and* Pascal
completeness 241–242 , 315, 322
vs. consistency 241, 325
see also first order predicate logic *and* resolution
computable language 1–3, 145, 152, 155, 158–159, 242, 245
TM-computable 242
see also under Turing machine
see also acceptable language, decidability, decidable language
conjunctive normal form 307–308, 312–314
see also conjunctive normal form database *and* resolution
conjunctive normal form database 312–314, 319–320
clause 312–322, 326
consistency of 315, 317–318
inference in 315–316
see also resolution
consistency 241
context free grammar 30–32, 33, 38, 42, 46, 77–78, 81–89, 92–98, 101–105, 113–116, 120, 122–130, 151, 158, 159, 267, 305, 327
empty string
ε production 83–84
removal of 81, 83–84, 89, 94, 105
non unit production 86
secondary production 87
unit production 76, 85–87
graph 85–86
removal of 85–86
see also context free language *and* push down recogniser
context free language 32, 81–82, 89–90, 97, 99–106, 112–118, 121, 128–130, 136, 141–143, 145, 333
closure properties 112–118
complement 117–118
implications 117–118
concatenation 115–116
intersection 116–117
union 113–115
deterministic 81, 100–101, 102, 104, 105, 113, 115–118, 122, 131, 140–141
closure properties
complement 118
concatenation 115–116
intersection 116–117
union 115
restricted palindromic language 102–103
non deterministic 81, 101–103, 104, 115, 117, 141–143
lack of equivalence with deterministic CFLs 81, 99, 101–102, 104, 105
proper 97
proper subset of type 0 languages 118
uvwxy theorem 107, 118, 122–130, 333
see also context free grammar *and* push down recogniser
context sensitive grammar 30, 129, 146, 150–152
context sensitive language 128–129, 145–146, 152, 233
see also under Turing machine

database, *see* conjunctive normal form database
de Morgan, Augustus 284
de Morgan's laws 110–111, 117, 284, 290, 337
see also propositional logic *and* set
decidability 155–157, 231, 234, 242, 245–247, 304, 316
decidable language 155–157, 158, 228, 242, 245
vs. acceptable 156–157
decidable problem 234
decision problem 234, 241
decision program 17–19, 30
compiler as 18–19
language definition by 17, 30, 32, 176, 289
relationship to grammars 32
sentence determination by 17–18
derivation 23, 26, 27–29, 33–34, 35–40, 45, 50, 87, 88, 122–129, 327
see also derivation tree

derivation tree 37, 38–45, 50, 58–59, 89, 122–128, 129, 329
- context free derivation 38–39
  - Chomsky normal form 89
- empty string in 38
- reading 38
- *see also* derivation

deterministic context free language, *see under* context free language
deterministic finite state recogniser (DFSR), *see under* finite state recogniser
deterministic pushdown recogniser (DPDR), *see under* pushdown recogniser
deterministic Turing machine, *see under* Turing machine
digital computer
- regarded as FST 176–178, 180
- state 177
- *see also under* finite state transducer

directed graph 85–86, 265–266, 298
- transitive closure 265

division by repeated subtraction 190, 192–193

effective procedure 204, 211, 232, 233–234, 241, 244, 289, 325
empty string 12–13
- *see also under* context free grammar, derivation tree, grammar, parsing, Pascal, pushdown recogniser, regular grammar, string concatenation

ε production, *see under* context free grammar *and* regular grammar
ε, *see* empty string
equivalence problem for regular languages 234
- *see also* decision problem

equivalence problem for Turing machines 241
- *see also* decision problem *and* halting problem

existential quantifier, *see under* first order predicate logic

finite state generator 51
- *see also* finite state recogniser

finite state recogniser (FSR) 49, 51–79, 82, 87, 89–90, 92–93, 96–97, 100–101, 103, 109–112, 118–121, 131, 133, 135, 139–140, 143, 145, 164, 234, 247, 265–268, 329–331
- acceptable string 56–61, 63–68, 93
- acceptance conditions 53–55
- adjacency matrix representation 265–266, 330, 336
- behaviour 51, 53–56
  - loop 118–119, 120, 333
- complement machine 108–109, 110–111, 131
- computational limitations of memory 89–90
- decision program 49, 61, 68–69
- deterministic (DFSR) 49, 61–76, 78–79, 96, 99–100, 108–112, 120, 131, 139–140, 163, 256, 330
  - linear time algorithm 256
- empty move 76–77, 82
- equivalence with regular languages 51, 56–59, 76–77, 89, 103, 140, 145, 234
- intersection machine 110–112
- list representation 331
- minimal 49, 68–76, 78, 234, 331
- non deterministic 59–68, 70–76, 78, 79, 96, 99, 251
  - equivalent to deterministic FSR 61–68, 78, 96, 234, 330
- rejection conditions 54
- reverse form 70–76, 331
- state 90
  - acceptance 53, 54
  - multiple start states 70, 77, 109–110
  - name 62, 70–73, 77
  - null 61, 68, 70, 100, 108
  - rejection 53–54
  - useless 267, 271, 336
- union machine 109–110

finite state transducer 3, 163–178, 179–180, 189, 234, 247, 250, 333–334
- computation 163–164, 167, 179
  - binary addition 168–172
  - binary subtraction 168–172
  - input preparation 167–168
  - limitations 163, 172–173
    - inability to perform arbitrary multiplication 173, 176, 179, 189
  - output interpretation 167–169
  - restricted division 171–172
    - number of states 176
  - restricted modular arithmetic 171–172
  - restricted multiplication 167–168, 172–176
    - number of states 176
- limited computational model 3, 176–178
- logical operation or 333
- memory 163, 250
  - shift operation 164–167
- no halt state 164
- regular language recogniser 164
- restricted TM 163, 179
- time 166–167, 250
  - clock 166–167

first order predicate logic (FOPL) 1, 4–5, 281, 291–306, 307–312, 313–314, 322, 323, 324–325, 326
- argument 292, 294–295, 316
- atom 294–295, 312
- classical reasoning 4–5, 301, 303–304, 305–306, 307, 317, 322, 326, 338
- completeness 304
- computational complexity 304–305, 307
- constant 292, 293, 294–295, 310, 314–316
- existential quantifier 291, 296–298, 299–300, 309–312, 315

first order predicate logic (FOPL) (*continued*)
function 291, 292, 293–294, 311–312, 313–314, 315–316, 324
arithmetic 294
nested call 293–294
interpretation (meaning) 292–294, 300–301, 309–310
validity 301, 304
monotonicity 305, 317, 322
predicate 291–296, 299–301, 306, 312–313, 316–320, 322–324
arithmetic comparison 293
vs. function 293–294
problem solving 291, 301, 303–304
quantified statement 296–299, 306
quantifier removal 312, 313
representation of time 299
rules of inference 299, 301–304, 308–309, 312–313
universal quantifier 291, 298–300, 301–303, 308–309, 314–316
variable 292, 294, 314, 315–316, 317
quantified 296–297, 303, 308–311, 314–316
scope 297
vs. type 0 language 304–305
well formed formula (sentence) 291, 294–295, 305
FOPL, *see* first order predicate logic
formal language 1, 2, 4, 6, 11, 14–22, 25, 37–38, 42, 108, 118, 131, 159, 242–243, 245–247, 247, 300
acceptability of 246–247
computational properties 30, 34
definition 14–22, 108, 131
see also set definition
finite 16
infinite 16, 112, 118, 120–121, 127, 128, 333
non computable 242–243
FSR, *see* finite state recogniser
FST, *see* finite state transducer
function 231–233, 249–250, 251
association between values 232, 334
(TM) computable 232–233
represented by program 232
vs. problem 232–233, 245–246
see also under first order predicate logic *and* set definition

Gödel Kurt 241, 325
Gödel's theorem 5, 241, 325
grammar 11, 22–36
(N, T, P, S) form 26–27
empty string in 26, 30
language generated by 26, 29–30
phrase structure 22–23, 26–27, 29, 30–31, 37, 45, 81, 82, 158
rule 19, 22
see also production
see also context free grammar, context sensitive grammar, regular grammar, unrestricted grammar
guard 54–55

halting problem 4, 178, 205, 231, 234–242, 245, 247, 304, 325, 334
human implications 242
linguistic implications 4
partial solvability of 4, 235, 240
programming implications 231, 234, 240–241, 247
reduction to 240–241, 247, 334
theoretical implications 240, 241–242
see also equivalence problem for Turing machines *and* printing problem for Turing machines
higher order logic 295–296

implication, *see under* propositional logic
infinite loop 117, 177, 234–240, 242
infix operator 171, 293, 294

Java [programming language] 330

k-tape Turing machine, *see under* Turing machine

lazy drunken students problem 304, 306, 307, 312–313, 314, 321–322, 323–324
leaf node 38
linear one dimensional array search 256–257, 260
linear two dimensional array search 261–262
average case 261–262
worst case 262
see also big O analysis, *clever* two dimensional array search, running time of algorithm
LISP [programming language] 12, 331
list 256, 330–331, 332
logarithm 258–260
base 10 259
base 2 259–260
estimation method 259–260
logic gate 278
modelling basic Boolean operators 279–280
NAND 278–280, 284, 290, 337
NOR 278–279, 284, 290, 337
see also Boolean logic
logical consequence 304, 319
logical implication, *see under* propositional logic
logical system
computational properties 271
linguistic properties 271
role in argumentation 271

see also Boolean Logic, first order predicate logic, propositional logic
LR(*k*) language 105
LR(1) 105

membership problem for language 233, 242–243, 245–246
monotonicity, *see under* first order predicate logic *and* resolution
multiplication language 104, 105, 129–130, 158
computational properties 34
not a context free language 104, 105, 129–130, 158
multiplication, shift and add method 180, 185–186, 188
base 10 form 185–186
binary form 185–186
multi-tape Turing machine, *see under* Turing machine

NAND gate, *see* logic gate
node 38, 86, 89, 126, 127, 129, 265–266
see also terminal node *and* leaf node
non determinism, computational implications 104, 270
non deterministic context free language, *see under* context free language
non deterministic finite state recogniser, *see under* finite state recogniser
non deterministic pushdown recogniser, *see under* pushdown recogniser
non deterministic Turing machine, *see under* Turing machine
non deterministic Turing machine, *see under* Turing machine
non terminal symbol 22–23, 26–28, 33–34, 35, 38, 43, 49, 58, 76, 81, 83–84, 85–89, 113, 122, 125–130, 145, 265, 327, 328, 333
NOR gate, *see* logic gate
NPDR, *see under* pushdown recogniser
octal (base 8) number system 178

palindromic string 101–103, 104–105, 141–143, 331
parallel computation 213, 249, 251–254, 268, 269, 288
implications 254
non deterministic Turing machine 251–252, 253–254, 268, 270
linear running time 268
polynomial running time 268, 269–270
see also running time of algorithm
see also under Turing machine
parsing 39–45, 49, 50–51, 60–61, 68, 77, 82, 84, 89, 105, 117, 146–150, 152–155, 288
aim of 40–41
bottom-up 39, 40–41
empty string
problems caused by 82, 84, 153–155,
look ahead 104–105
reduction method 41–42, 50, 89, 146–151, 152–155
Chomsky normal form 89
reductions vs. productions 41, 89
top-down 40–41
Pascal 2, 12, 19–22, 37, 42–45, 68, 77–78, 82, 89, 121, 145, 180, 328, 329, 330
arithmetic expression 43–44
begin..end construct 78, 121
Boolean expression 43–44
case statement 82
char data type 68
compiler error 145
compound statement 44
defined by syntax diagrams 19–22
defined in Backus-Naur form 20–22, 328
empty string 82
compilation 82
function 180
identifier 19–21, 77, 328
if statement 82
non regular language 78, 121
procedure 180
program construct 19–21
record data type 89
source code 207
scope of variable 297
PDR, *see* pushdown recogniser
phrase structure language 30, 145, 155, 242, 246
see also phrase structure grammar
$P = NP$ problem 269–270
unsolved 249, 254, 270
see also decision problem, *see also under* Turing machine
pop, *see under* push down recogniser
Post, Emile
abstract machine theory 204
predicate logic, *see* first order predicate logic
printing problem for Turing machine 247
see also decision problem, halting problem
problem 231–232
question associated with function 232–233
see also decidable problem, decision problem, halting problem, membership problem, $P = NP$ problem, semidecidable problem, solvable problem, unsolvable problem
production 22–23, 25–32, 38, 41, 49, 58, 61, 76–77, 81, 82–89, 94, 145–148, 150–155, 267, 288, 327, 328, 329
program correctness
lack of algorithmic solution 240
partial 240
programming language 1–2, 12, 18–19, 20–22, 32, 37, 42–43, 44–45, 77–78, 82, 84, 104, 117, 118, 121, 171, 205, 279, 297, 324
statement 15
if statement 15
implications of non deterministic CFLs 104

programming language (*continued*)
integer division (*div*) 172, 173
modulus (*mod*) 172
scope of variable 297
source code 37, 40, 82
subroutine 180
syntax definition 19, 20–22
unconditional jump (*goto*) 68
vs. natural language 20, 40, 45
PROLOG 5, 12, 307, 323–325, 331
backtracking mechanism 323
program execution 323–324
vs. other programming languages 324–325
propositional logic 1, 4, 275, 281–289, 290, 291–292, 299, 301, 305
computational complexity 286, 288–289
connective 281
consistency 288
equivalence operator 283
role in reasoning 284
expressive limitations 4, 289, 291
implication operator 4, 282–285, 303, 314
necessary condition 282–283, 323
sufficient condition 282–285
interpretation (meaning) 281
problem solving 284, 285–288
proposition 281–282, 283, 289, 291, 296
atomic 281, 285, 287, 289, 291
rules of inference 5, 282, 284–286, 288, 290, 299, 301–302
de Morgan's laws 284, 313, 321
modus ponens 285, 287, 303, 323
modus tollens 285
theorem 287
PSG, *see* phrase structure grammar
PSL, *see* phrase structure language
push, *see under* push down recogniser
pushdown recogniser (PDR) 81, 90, 92–105, 106, 113, 115, 117–118, 122, 130, 133, 136, 141, 143, 152, 158, 331–332
computational limitations of 103, 130, 158
deterministic (DPDR) 81, 96–103, 104, 105, 106, 114, 332
acceptance conditions 101
difference from NPDR 101
behaviour 97–98, 106
characterisation of 97
modelling DFSR 100
unable to clear stack 331
equivalence with context free languages 81, 103–104, 141, 145
non deterministic (NPDR) 81, 92–96, 97, 99, 101, 103–104, 105, 106, 117, 133, 141, 143, 152
acceptance conditions 93
behaviour 94
characterisation of 94
lack of equivalence with DPDRs 97, 99, 101–104
quintuple representation 332
relation to FSR 90
simulation by program 332
stack 81, 90–94, 95–96, 104, 105, 141, 247, 331, 332
bottom 90, 101
operation
pop 90–91, 97
push 90–91, 97, 102
empty string 90, 91
top 90, 94, 97

queue 90
relationship to stack 90
quicksort 260, 261, 262, 271
pivot 260, 272
vs. $O(n^2)$ algorithm 260

recursive definition 159, 294
well formed parentheses 159
well formed formula, *see under* first order predicate logic
recursive function theory 204
reductio ad absurdum 117, 121, 128, 130, 176, 307, 320
reduction parsing, *see under* parsing
reflexive relation 306
regular expression
equivalence with regular languages 77
in text editor 77
regular grammar 35, 36, 38, 42, 49–51, 56–60, 68, 70, 76–78, 84, 87, 89, 100, 103, 112, 113, 121, 145, 234, 271, 327–328
empty string 84
ε production 84
removal 84
useless production 267
see also finite state recogniser, regular expression, regular language
regular language 35, 36, 49, 50, 64, 68, 77–78, 81, 97, 99–101, 103, 104, 107–112, 116, 117, 118, 122, 128, 131, 140, 145, 164, 247, 329
as deterministic CFL 99–100
finite 36, 329
proper subset of CFLs 104, 120–122
closure properties
complement 108–109
concatenation 112, 113
intersection 110–112, 117
union 107, 109–110, 111
repeat state theorem for FSRs 107, 118–122, 128, 131, 176, 177, 179, 203, 333
resolution 5, 305, 307, 314, 317
application 316, 322, 338–339
completeness 322
computational complexity 322
justification 322–323
monotonicity 317, 322
resolvent 317, 320, 322
set of support strategy 322
unit preference strategy 322

running time of algorithm 4, 249, 250–251, 255–265, 267–268, 270–272, 288–289, 336
 average case 256
 best case 256
 dominant process 255
 exponential 249, 256, 267–270, 288
  implications 268–270
 implications 270
 linear 249, 255, 256–257, 260, 263, 264, 268, 271, 272, 336
 logarithmic 249, 255–256, 258–259, 260, 261
  vs. linear 260
 optimal 255, 270
 polynomial 249, 255, 256, 260–267, 268, 269–270, 336
 worst case 256, 261–262, 264

search algorithm, *see* binary search, *clever* two dimensional array search, linear one dimensional array search, linear two dimensional array search,
self reference 325
 in logical systems 241
 in Turing machines 241, 325
semantics 20, 37–38, 46, 47, 300, 308, 325, 329
 of programs 20, 37
 vs. syntax 20, 37–38
semidecidability of first order predicate logic 304–305
semidecidable language 231, 245
semidecidable problem 304
sentence 11, 15, 17–19, 22, 24, 28–29, 33–34, 37–38, 40–41, 45, 58, 117, 122, 130, 155–156, 233, 242, 245
 word as synonym for 15
 see also sentential form *and* terminal string
sentence symbol, *see* start symbol
sentential form 28–29, 33–34, 38, 89, 304–305
sequence 256
sequential search, *see* linear one dimensional array search and linear two dimensional array search
serial computation
 deterministic 4-tape machine 252–254, 268
  exponential running time 268–269
  polynomial running time 270
  vs. parallel 249, 251, 252–254, 268–270
   see also under Turing machine
set definition (of formal language) 15–17, 19, 29–30, 35, 50, 81, 85
set
 complement 108–109, 110–112, 117–118, 131
 concatenation 112, 115–116
 de Morgan's law 110–111, 117, 284, 290, 337
 difference 17, 99, 108
 empty 14, 267
 enumerable (countable) 14, 217
  systematic generation (by TM) 216–222, 229, 243–244,
 finite 328
 infinite 108
 intersection 17, 107, 110, 116–117
 subset 58, 108
  non proper 14
  proper 14, 30, 118, 122, 131
 union 17, 107, 109–110, 113–115
Shannon, Claude 250
Skolem, A. T 312
Skolemisation 312
 Skolem constant 312
 Skolem function 312, 315
 see also first order predicate logic
solvability 231, 232–234, 242, 246
 partial 233
 total 233, 234
 vs. decidability 233, 246
 see also problem
solvable problem 233, 246
sort algorithm, *see* bubble (exchange) sort *and* quicksort
space requirement 249, 251
 of program 249–250
 of Turing machine 250
stack, *see under* push down recogniser, *see also* queue
start symbol 22–23, 25–29, 38, 41, 58, 84, 86, 89, 94, 147, 151
string (dynamic data structure) 12–13
string (formal) 12–14
 defining properties 12
see also empty string, palindromic string, string concatenation, string index operator, string power operator
string concatenation 12, 16, 82, 112, 115
 empty string in 12
string index operator 12, 16, 58, 68
string power operator 12, 16, 112
sub-machine TM 180, 182, 190, 227
 ADD 180–190, 192–193, 197, 198, 199, 227, 228
 COMPARE 194–196, 197, 199, 232
 COPY-L 187–188
 COPY-R 188
 INVERT 192
 SUBTRACT 190–193, 197, 199, 227
 TWOS-COMP 192
 WRITE-A-ONE 180–181, 182–183
 WRITE-A-ZERO 180, 181, 183
 see also Turing machine
subset construction algorithm for FSRs 62–65, 70–76, 78–79, 109, 267–268, 271, 330, 336
 exponential time 267–268
 termination condition 79
 worst case 267–268, 336
 see also big O analysis, finite state recogniser, running time of algorithm
substring 23, 41, 119, 121, 123, 125–129, 147
 see also string (formal) *and* empty string
subtraction by addition 191–192

symmetric relation 306
syntax 2, 19–22, 32, 34, 37, 44, 78, 82, 117, 197, 294, 300, 308, 325
syntax diagram19–22
  see also Backus-Naur form *and* Pascal

terminal node 38, 58, 89, 127
terminal string 24–26, 28, 29, 40–41, 86–88, 115, 123, 147, 155, 265, 327
  see also sentence
terminal symbol 22, 23–24, 26–29, 38
TM, *see* Turing machine
transitive relation 298, 303, 326
truth table 4, 275–278, 282, 283, 286–289, 290, 337–338
  as decision program 289
  as Turing computable function 289
    see also function
  exponential nature 289
  statement decomposition 267
Turing machine 2–4, 34, 130, 133–159, 163, 177, 179–202, 203–229, 231–247, 249–254, 269–270, 289, 305, 325, 333, 334–335
  abstraction over real machines 205
  architecture
    alphabet 134, 136
    blank 133–136, 137, 138, 139–141, 148–150, 151, 152–153, 163, 180–181, 187, 188, 206, 216, 217, 222, 223, 226. 233, 235, 333
    read/write head 133–135, 136, 139, 141, 180, 186, 203, 214, 222–227, 233, 235, 251, 333, 335
    tape 4, 133–144, 151–152, 155–156, 158, 163, 177, 180–190, 193–195, 200–201, 204, 205, 207–229, 233, 240, 241, 250–252, 254, 268, 333, 334–335
      packed 148, 151, 216, 233
      square 133–135, 137, 148, 149, 151, 153, 193, 214, 215, 216, 223, 226–227, 233, 250
  behaviour 4, 135–139
    erasing 188, 193, 204, 215, 223, 235, 335
    moving 133, 136
    reading 133
    scanning 148, 151
    shuffling 137, 148–149, 151, 152, 153, 188, 193–194, 226–227
    skipping 138, 151
    ticking off 142, 188, 218
    writing 134
  blank tape assumption 140, 141, 155
  coding of 4, 203, 232, 235, 239, 243–245
    direction symbols 206–207
    input tape 205–206, 208–209
    machine 203, 205, 208
  complexity
    number of states vs. number of symbols 250–251
  computation 158–159, 179, 199
    arbitrary integer division 190, 193–194
      DIV 190, 193–196, 199, 227, 232
    arbitrary multiplication 180, 185–190
      MULT 185–190, 193, 199, 207, 227–228, 232
    binary addition 180–186, 192–193, 194, 199
    binary subtraction 191–193
    most powerful device 177, 203, 247
    power exceeds computer 3–4, 159, 177, 179–180, 196–199, 205, 249–250
    vs. FST 177, 178
  configuration
    input 138–139, 143, 148, 155, 182, 186, 187, 194, 200, 201, 253, 268
    output 139, 143, 194
  deterministic 243–244
  function 200
  instantaneous description 251
  language processing 133, 139, 247
    equivalence with type 0 languages 158, 228
    palindromic language 141–143
    type 0 language recogniser 130–131, 133, 145, 152, 158, 228, 242, 245–247
    type 1 language recogniser 130, 146, 152, 246–247
    vs. FSR 133, 140–141, 155, 247
    vs. PDR 133, 141–144, 155, 247
  logical operations 179, 197, 199
    comparison of two numbers 194–196
  model of computer 179–180
  multi-tape (k-tape) version 4, 203, 208–209, 213–214, 222, 226–228
    4-tape example 214–216, 218–222, 226, 252
    convenience 227–228
    equivalence with single-tape version 203, 209, 211, 213–214, 222, 227–228
    sextuple representation 222, 226, 229, 335
    simulation by 1-tape TM 222, 226, 229
      coding 3 tapes onto 1 223, 225–226
    simulation of non deterministic TM 214–222, 252–254
  non deterministic 147–148, 152, 155, 203, 213, 222, 228, 242, 251, 254, 268, 269–270
    equivalence with deterministic k-tape TM 203, 213, 222, 228
    equivalence with deterministic TMs 152, 213, 228, 254, 269–270
    problem solving process 213
  quintuple representation 206–207, 208, 210–213, 215–218, 222–224, 226, 228, 235, 240, 241, 244, 252–254, 268–269
  simulation of computer 196–197
    memory & array access 179, 199, 200, 250
    program execution 198, 199
  state
    halt state 138–139
      single 140, 159, 333
Turing machine simulation program 228–229

Turing, Alan 2, 133, 204
Turing's thesis 4, 5, 193, 203–204, 211, 213, 217, 231, 243, 250, 324, 325
  evidence in support of 204
  vs. theorem 203–204, 325
twos complement number 191–192
type 1 grammar, *see* context sensitive grammar
type 1 language, *see* context sensitive language

unacceptable (non computable) language 243–244
undecidable language 231, 245
unification 307, 314, 315–316, 320, 323
  of predicates 316–317, 318, 320, 322
  rules of 315–316
  substitution 314–317
  totally decidable 316
  see also conjunctive normal form database *and* first order predicate logic
unit production graph, *see* context free grammar
unit production, *see* context free grammar
universal logic gate, *see* logic gate
universal quantifier, *see under* first order predicate logic
universal Turing machine 3, 203, 205, 207, 208–213, 228, 232, 235, 244
  3-tape version 203, 208–212
  behaviour 210–212, 235
  implications 211, 213
  input configuration 209–211
  model of stored program computer 211
  partially solves halting problem 234–240, 245
  simulation of TM 208–211, 213
  see also Turing machine *and* halting problem
unrestricted grammar 32–34, 35, 130, 145–146, 152–155, 158–159, 213
  see unrestricted language
unrestricted language 118, 130–131, 158, 203, 213, 228, 247, 304
  see Turing machine *and* unrestricted grammar
unsolvability 231, 233, 234–235, 239–242, 247
  total 233, 247
unsolvable problem 205, 325
useless production, *see under* regular grammar
useless state, *see under* finite state recogniser
uvwxy theorem for CFLs *See Under* Context Free Language

Wang machines
  vs. TMs 204
Warshall's algorithm 265–267, 271
  applied to adjacency matrix 265–267
    connectivity matrix 266
      see also finite state recognisers