Some error correcting codes for certain transposition and transcription errors in decimal integers

D. A. H. Brown

Royal Radar Establishment, St. Andrews Road, Malvern, Worcs. WR14 3PS

The standard theory of modulus 11 cyclic block error-correcting codes is applied to numbers expressed in the decimal system. An algorithm for error correction is given.

(Received December 1972)

1. Introduction

In the copying of decimal integers Wild [1968] has classified the common types of error as:

(*a*) Transcription errors which can be subdivided into:
Single: only one digit is in error, type $a \to b$, and
Multiple: more than one digit is in error. An important sub-class is where several repeated digits are incorrectly copied in the same manner as in Wild's example of 7655597 being copied as 7688897, type $aaa \to bbb$.

(*b*) Transposition errors: types such as $ab \to ba$, $axb \to bxa$ $axyb \to bxya$.

(*c*) Shift errors: a repeated digit is repeated either too many times (left shift error) or too few times (right shift error).

(*d*) Random errors: Wild defines these as 'errors not specifically guarded against by the choice of weight and N'. (N is the modulus check integer e.g. 11 in a modulus 11 system.) These would be better called undetectable errors (by the code in use). In principle multiple transcription errors of no special type are random errors.

Much effort has been devoted to empirical modulus 11 systems and a suggested modulus 97 system (Wild, 1968; Briggs, 1970; Reid, 1970; Beckley, 1967; Briggs, 1971) to detect errors of the main types above. This paper considers some error correcting codes for certain of these errors. The codes are also based on modulus 11 so that they suffer from the well known restriction applicable to all such systems that, in general, the check digits can take the value 10 and the representation of this value requires an extra symbol in addition to the decimal digits 0, 1 . . . 9. True decimal error detection systems, in which the check digits take only the values 0 to 9, have been studied by Campbell (1970) and Andrew (1970; 1972) but, as Bell (1972) has pointed out, true decimal error correction is nowhere in sight.

The code principles and method of error correction are first explained by reference to a specific code of 19 digits of which 16 are data and 3 are check digits which will correct errors of the following types: (*a*) single transcription error $a \to b$; (*b*) a double repeated transcription error, $aa \to bb$; (*c*) two types of transposition error $ab \to ba$ and $axb \to bxa$. The correction capability $ab \to ba$ can alternatively be exchanged for the ability to correct a length 3 multiple transcription error, $aaa \to bbb$. Finally some other codes are discussed including a (10, 6) double error correcting code. The notation (n, k) refers to a code with a total block length of n digits of which k are data digits and $n - k$ are check digits.

The theory of block cyclic codes based on polynomials having coefficients in a finite field $GF(q)$ where q is a prime number is available in standard texts (Peterson and Weldon, 1972; Berlekamp, 1968) and no review of this material is attempted here. Tang and Lum (1970) have given general results on the application of polynomial codes to correct transposition errors.

2. Code formation

The (19, 16) code to be discussed is based on the polynomial $X^3 - 10X^2 - 9X - 1$ which has coefficients in $GF(11)$, the field of integers modulus 11, and is irreducible in that field. Irreducible means that it has no factors of lower degree when the coefficients are restricted to the integers 0, 1 . . . 10. The polynomial is associated with the transition matrix

$$T = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 9 \\ 0 & 1 & 10 \end{bmatrix}$$

and this is used to generate the weighting matrix H for the code namely

$$\begin{array}{cccccccccccccccccccc} & d_0 & & d_2 & & d_4 & & d_6 & & d_8 & & d_{10} & & d_{12} & & d_{14} & & d_{16} & & d_{18} \end{array}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 10 & 10 & 4 & 8 & 5 & 5 & 4 & 2 & 6 & 5 & 7 & 0 & 2 & 5 & 2 \\ 0 & 1 & 0 & 9 & 3 & 1 & 2 & 10 & 9 & 6 & 8 & 0 & 1 & 7 & 2 & 7 & 7 & 3 & 1 \\ 0 & 0 & 1 & 10 & 10 & 4 & 8 & 5 & 5 & 4 & 2 & 6 & 5 & 7 & 0 & 2 & 5 & 2 & 1 \end{bmatrix}$$

The matrix T operates on each column of H to form the next column to the right and it is seen that the sequence so formed is of length 19 which gives the code length. (In the theory of the finite field this is the order of the root α of the polynomial.) A code vector v consists of 19 digits forming a row vector $[d_0 d_1 d_2 d_3 \ldots d_{17} d_{18}]$ where $d_0\ d_1\ d_2$ are the check digits and d_3 to d_{18} the data digits with d_{18} as most significant. A code vector has the check digits chosen so that $vH^T = 0$. Check digit d_0 corresponds to the weights in row 1 of H, d_1 to row 2 and d_2 to row 3.

An integer is checked by forming the check sums $c_0 c_1 c_2$ corresponding to the row weights and if $c_0 c_1 c_2$ are not all zero then an error has occurred. The error may have one of the correctable patterns or it may have some other form and be detectable as an error but not correctable.

3. Error correction

The three check sums form a column vector $\{c_0 c_1 c_2\}$. The correction algorithm is as follows, written in a form akin to Algol 60.

comment *first set the variables* $z_0 z_1 z_2$ *to the initial values of the check digits*;
$z_0 := c_0;\ z_1 := c_1;\ z_2 := c_2;$
comment *initialise a pointer p to the code length, here 19*;
$p := 19;$
for $p := p - 1$ **while** $p \geqslant 0$ **do**
begin comment *at each repetition of the for loop first form the product* $[T][Z]$ *where* $[T]$ *is the transition matrix and* $[Z]$ *is the column vector of the variables* $\{z_0 z_1 z_2\}$ *which take different values at each repetition.* y_0, y_1, y_2 *are temporary stores used during the matrix multiplication*;
$y_0 := \quad\quad z_2;$
$y_1 := z_0 + \ \ 9z_2;$
$y_2 := z_1 + 10z_2;$
$z_0 := y_0;\ z_1 := y_1;\ z_2 := y_2;$
if $z_1 = 0$ **and** $z_2 = 0$
then
begin comment *a single error has occurred, the correction*

to digit d_p *is*;

$d_p := d_p - z_0$; **go to** *success*

end

else if $z_0 = z_1$ **and** $z_2 = 0$

then

begin comment *a double repeated error of the type* $aa \rightarrow bb$ *has occurred, the correction is*;

$d_p := d_p - z_0$; $d_{p+1} := d_{p+1} - z_0$; **go to** *success*

end

else if $z_0 + z_1 = 0\ (mod\ 11)$

then

begin comment *a transposition error in adjacent digits has occurred of the type* $ab \rightarrow ba$, *the correction is*;

$d_p := d_p - z_0$; $d_{p+1} := d_{p+1} - z_1$; **go to** *success*

end

else if $z_1 = 0$ **and** $z_0 + z_2 = 0\ (mod\ 11)$

then

begin comment *a transposition error of the type* $axb \rightarrow bxa$ *has occurred, the correction is*;

$d_p := d_p - z_0$; $d_{p+2} := d_{p+2} - z_2$; **go to** *success*

end

end *for loop*;

fail: **comment** *if the program comes to this point then all* 19 *positions have been tried without encountering a correctable error pattern. The error is not correctable*;

success: **comment** *a jump to this point indicates that correction has taken place*;

4. Theoretical basis

Irreducible polynomials with coefficients in a field of integers modulo a prime number, here 11, are closely associated with certain cycles. One such cycle is seen in the formation of the weighting matrix H where the transition matrix operating on the final column {211} gives the starting column {100}. There are 10 such cycles starting {200}, {300} etc., and the columns of the cycle starting $\{n00\}$ are all n times (modulo 11) those in H; each is of length 19. Suppose now that a single error occurs in the digit d_7 corresponding to the column {8, 10, 5} and that the error is an increase in this digit of magnitude +3. Then $\{c_0c_1c_2\} = 3\{8, 10, 5\} \equiv \{2, 8, 4\}$ modulo 11. If we now operate on this successively with T we obtain

start	d_{18}		d_{16}		d_{14}		d_{12}		d_{10}		d_8	d_7
2	4	4	1	6	7	4	10	0	6	4	6	3
8	5	7	2	0	3	10	6	10	10	9	3	0
4	4	1	6	7	4	10	0	6	4	6	3	0

When the correctable error pattern {300} is reached the pointer $p = 7$ indicating that digit 7 is in error by +3.

These 10 cycles of length 19 use up 190 of the possible error patterns. In the same way 10 other cycles of length 19 start with the vectors {110}, {220}, etc. and correspond to two equal adjacent digits being replaced by an incorrect but equal pair, $aa \rightarrow bb$. The cycle starting {110} is

1	0	1	0	9	3	1	2	10	9	6	8	0	1	7	2	7	7	3
1	1	9	1	4	3	1	8	4	3	8	1	8	9	9	3	10	4	1
0	1	0	9	3	1	2	10	9	6	8	0	1	7	2	7	7	3	1

and it is important to note that this cycle nowhere has on it a vector of the form $\{n00\}$ which would correspond to one of the single error cycles.

Similarly the error corresponding to transposition of adjacent digits uses up another 10 cycles of length 19 starting {1, 10, 0}, {2, 9, 0}, {3, 8, 0} where the two non zero values always add to zero modulo 11. This is because the transposition of two adjacent digits corresponds to adding an error e to one digit and its complement modulo 11 to the other e.g. 69 reversed gives 96 which corresponds to adding 3 to the first (correct) digit 6 and adding $11 - 3 = 8$ to the second so that

$$9 + 8 = 17 \equiv 6 \bmod 11\,.$$

The cycle starting {1, 10, 0} is

1	0	10	2	0	6	7	3	0	1	2	7	1	9	7	9	8	3	1
10	1	2	6	2	10	3	1	3	9	8	10	5	5	6	0	4	2	1
0	10	2	0	6	7	3	0	1	2	7	1	9	7	9	8	3	1	1

and it is particularly important to note that this contains the vector {111} which corresponds to an error $aaa \rightarrow bbb$. This means that the code can be used to correct two alternative error patterns, either $aaa \rightarrow bbb$ or $ab \rightarrow ba$ but not both. If one is chosen for correction then if the other occurs it will be falsely 'corrected'; this could be dangerous unless it is known from error statistics that the probability of occurrence of the unchosen error type is negligibly small.

Finally the cycles corresponding to the error $axb \rightarrow bxa$ start {1, 0, 10} {2, 0, 9} etc. That starting {1, 0, 10} is

1	10	1	2	6	2	10	3	1	3	9	8	10	5	5	6	0	4	2
0	3	8	8	1	2	4	4	1	6	7	4	10	0	6	4	6	3	0
10	1	2	6	2	10	3	1	3	9	8	10	5	5	6	0	4	2	1

The above system uses up $4 \times 190 = 760$ error combinations out of the 1330 $(= 11^3 - 1)$ available. This means that if errors occur with low probability in blocks of coded digits and these errors are 'random' ones there is a one in 1331 chance the error will go undetected, a chance 570/1330 that it will fall into the category of an uncorrectable error and a chance 760/1330 that the error correction procedure will be invoked. This will normally restore the original (correct) number but it includes cases having a very low probability of occurrence such that the error is falsely 'corrected', that is the number finally given by the error correction procedure is not the original number.

5. Other codes similarly implemented

There are many codes which can be implemented in a similar way, giving various (n, k) combinations and error detection/correction capability. The best code to be used in given circumstances can be designed around the error statistics once these are known. Some examples of other codes are:

5.1. $(X^2 - 2)(X - 2) \equiv X^3 - 2X^2 - 2X - 7$

$$T = \begin{bmatrix} 0 & 0 & 7 \\ 1 & 0 & 2 \\ 0 & 1 & 2 \end{bmatrix} \quad (n, k) = (20,17)$$

The cycle beginning {100} corrects a single error, {110} corrects $aa \rightarrow bb$, {111} corrects $aaa \rightarrow bbb$ and {1, 10, 0} corrects $ab \rightarrow ba$. These corrections use 800 out of the 1330 error combinations.

5.2. $(X - 2)(X - 4)(X - 8)(X - 5) \equiv X^4 - 8X^3 - 6X^2 - 3X - 10$

$$T = \begin{bmatrix} 0 & 0 & 0 & 10 \\ 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & 1 & 8 \end{bmatrix} \quad (n, k) = (10, 6)$$

This is a double error correcting Bose-Chaudhuri-Hocquenghem code. The mechanics of double error correction are given in Section 6. If not used for double error correction it can be used to correct the following special errors: (*a*) {1000} for single errors; (*b*) {1100}, {1110}, {1111} for $aa \rightarrow bb$, $aaa \rightarrow bbb$ and $aaaa \rightarrow bbbb$ respectively; (*c*) {1, 10, 0, 0}, {1, 0, 10, 0}, {1, 0, 0, 10} for transposition errors $ab \rightarrow ba$, $axb \rightarrow bxa$ and $axyb \rightarrow bxya$. These corrections use only 700 out of the $11^4 - 1$ error combinations so that many other errors will be detected.

5.3. $(X^2 - 10X - 10)(X - 2) \equiv X^3 - X^2 - X - 2$

$$T = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad (n, k) = (30, 27)$$

This is a good example of how dangerous error correction can be. The single error correction cycle starting {100} includes on it the patterns {10, 10, 0} and {10, 0, 10} corresponding to the errors $aa \to bb$ and $axa \to bxb$. The correction cycle for $ab \to ba$ starting {1, 10, 0} includes also the pattern {10, 0, 1} corresponding to $axb \to bxa$. It would therefore be very dangerous to use the code for correcting single errors or adjacent digit transposition errors unless the other types of error were known to be very improbable.

6. Random double error correction

No attempt is made here to explain the full theory of the Bose-Chaudhuri-Hocquenghem (BCH) codes for random multiple errors. This is available in the standard texts (Peterson and Weldon, 1972; Berlekamp, 1968). All that is attempted here is to demonstrate the code construction and the actual mechanics of error correction which are not too difficult in this case. The code used is the (10, 6) code defined by $X^4 - 8X^3 - 6X^2 - 3X - 10$ of Section 5.2. It is associated with the transition matrix given in Section 5.2 and the weighting matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 10 & 3 & 7 & 5 & 4 & 3 \\ 0 & 1 & 0 & 0 & 3 & 1 & 4 & 3 & 4 & 6 \\ 0 & 0 & 1 & 0 & 6 & 7 & 3 & 7 & 1 & 8 \\ 0 & 0 & 0 & 1 & 8 & 4 & 6 & 7 & 8 & 10 \end{bmatrix} \qquad (1)$$

An alternative form of weighting matrix is obtained by row operations on (1) so that

$$\begin{bmatrix} 1 & 2 & 4 & 8 \\ 1 & 4 & 5 & 9 \\ 1 & 8 & 9 & 6 \\ 1 & 5 & 3 & 4 \end{bmatrix} \begin{bmatrix} \\ \text{Matrix 1} \\ \\ \end{bmatrix} = \begin{bmatrix} 1 & 2 & 4 & 8 & 5 & 10 & 9 & 7 & 3 & 6 \\ 1 & 4 & 5 & 9 & 3 & 1 & 4 & 5 & 9 & 3 \\ 1 & 8 & 9 & 6 & 4 & 10 & 3 & 2 & 5 & 7 \\ 1 & 5 & 3 & 4 & 9 & 1 & 5 & 3 & 4 & 9 \end{bmatrix} \qquad (2)$$

This shows the basic construction of the weighting matrix of the code. In the form (2) row 1 consists of the powers of 2 mod 11, row 2 the powers of 2^2, row 3 the powers of 2^3, row 4 the powers of 2^4. 2 is a primitive element of the field of integers modulus 11, that is its powers exhaust the field. The four rows give four check sums which enable the four unknowns, two error locations and two error magnitudes to be found.

If we use the weighting matrix in the form (1) and assume errors in a coded integer of magnitude +7 in d_4 and +9 in d_8 then the check sums will be

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = 7 \begin{bmatrix} 10 \\ 3 \\ 6 \\ 8 \end{bmatrix} + 9 \begin{bmatrix} 4 \\ 4 \\ 1 \\ 8 \end{bmatrix} = \begin{bmatrix} 4 \\ 10 \\ 9 \\ 1 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \\ 9 \\ 6 \end{bmatrix} = \begin{bmatrix} 7 \\ 2 \\ 7 \\ 7 \end{bmatrix}$$

First these must be converted to the other form by

$$\begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 4 & 8 \\ 1 & 4 & 5 & 9 \\ 1 & 8 & 9 & 6 \\ 1 & 5 & 3 & 4 \end{bmatrix} \begin{bmatrix} 7 \\ 2 \\ 7 \\ 7 \end{bmatrix} = \begin{bmatrix} 7 \\ 3 \\ 7 \\ 0 \end{bmatrix}$$

Now assume that the error magnitude and locations are unknown but that we have found $S_1\ S_2\ S_3\ S_4$ from the check sum tests on a received (incorrect) number. Let the error magnitudes be a and b and the error locations X_1 and X_2 where X_1 and X_2 are values in the first row of matrix (2).

Then using the matrix form (2) and its four rows we obtain

$$\begin{aligned} aX_1 + bX_2 &= S_1 \\ aX_1^2 + bX_2^2 &= S_2 \\ aX_1^3 + bX_2^3 &= S_3 \\ aX_1^4 + bX_2^4 &= S_4 \end{aligned}$$

The usual method of solving these equations is to assume an error location polynomial

$$X^2 + \sigma_1 X + \sigma_2 = 0$$

which has as its roots X_1 and X_2, the error locations, and then to find σ_1 and σ_2 in terms of $S_1\ S_2\ S_3\ S_4$. This is easily done. Because the error location polynomial has as its roots X_1 and X_2 we have

$$\begin{aligned} X_1^2 + \sigma_1 X_1 + \sigma_2 &= 0 \\ X_2^2 + \sigma_1 X_2 + \sigma_2 &= 0 \end{aligned}$$

If we multiply the first of these by aX_1^2 and the second by bX_2^2 and add we obtain

$$S_4 + \sigma_1 S_3 + \sigma_2 S_2 = 0$$

Similarly if the multipliers are aX_1 and bX_2 we obtain by addition

$$S_3 + \sigma_1 S_2 + \sigma_2 S_1 = 0$$

We then have two simultaneous equations for σ_1 and σ_2 which give

$$\sigma_1 = \frac{S_2 S_3 - S_1 S_4}{S_1 S_3 - S_2^2} \qquad \sigma_2 = \frac{S_2 S_4 - S_3^2}{S_1 S_3 - S_2^2}$$

Using the values for $S_1 S_2 S_3 S_4$ which arise from the assumed errors we obtain

$$\sigma_1 = \frac{21 - 0}{49 - 9} = \frac{21}{40} \equiv \frac{21}{7} = 3$$

$$\sigma_2 = \frac{0 - 49}{40} = \frac{-49}{40} = -7 \equiv +4$$

We then have to find the roots of $X^2 + 3X + 4 = 0$. The roots are 3 and 5. In the top row of matrix (2) the values 5 and 3 correspond to the powers 2^4 and 2^8 modulo 11 so that digits 4 and 8 are in error and we have found the error locations. It is now easy to find the error magnitudes. From the first two rows of matrix (2) we obtain

$$\begin{aligned} a.5 + b.3 &= S_1 = 7 \\ a.3 + b.9 &= S_2 = 3 \end{aligned}$$

which give $a = 7$ and $b = 9$.

If only a single error has occurred the denominator of the expressions for σ_1 and σ_2 namely $S_1 S_3 - S_2^2$ will be zero. Although the expressions are then indeterminate it is easy to test for this special case.

7. Conclusion

An attempt has been made to show that modulus 11 error detection and correction codes for decimal integers are easily constructed and implemented using polynomial theory. Hopefully it may stimulate others to investigate and use them.

Acknowledgements

I wish to thank the referee for several helpful comments which have been incorporated in the paper and particularly for his insistence that the codes are not true decimal codes (symbols confined to 0 to 9) but codes modulus 11. He also drew my attention to the paper by Tang and Lum.

References

ANDREW, A. M. (1970). A Variant of Modulus 11 Checking. *The Computer Bulletin*, Vol. 14, No. 8, pp. 261-265.

ANDREW, A. M. (1972). Decimal Numbers with Two Check Digits. *The Computer Bulletin*, Vol. 16, No. 3, pp. 156-159.

Beckley, D. F. (1967). An Optimum System with Modulus 11. *The Computer Bulletin*, Vol. 11, No. 3, pp. 213-215.
Bell, D. A. (1972). Decimal Numbers. *The Computer Bulletin*, Vol. 16, No. 8, p. 373.
Berlekamp, E. R. (1968). *Algebraic Coding Theory*. McGraw Hill.
Briggs, T. (1970). Modulus 11 Check Digit Systems. *The Computer Bulletin*, Vol. 14, No. 8, pp. 266-269.
Briggs, T. (1971). Weights for a Modulus 97 System. *The Computer Bulletin*, Vol. 15, No. 2, p. 79.
Campbell, D. V. A. (1970). A Modulus 11 Check Digit System for a Given System of Codes. *The Computer Bulletin*, Vol. 14, No. 1, pp. 12-13.
Peterson, W. W., and Weldon, E. J. (1972). *Error Correcting Codes* (2nd Edition). MIT Press.
Reid, C. J. (1970). Modulus 11 Check Digits. *The Computer Bulletin*, Vol. 14, No. 4, p. 122.
Tang, D. T., and Lum, V. Y. (1970). Error Control for Terminals with Human Operators. *IBM Jour Res and Devel.* Vol. 14, No. 4, pp. 409-416.
Wild, W. G. (1968). Theory of Modulus N Check Digit Systems. *The Computer Bulletin*, Vol. 12, No. 12, pp. 309-311.

Book review

The Theory of Parsing, Translation and Compiling, by Alfred V. Aho and Jeffrey D. Ullman; Volume 1: Parsing, 1972, 541 pages, £8·75, Volume 2: Compiling, 1973, 460 pages, £8·60. (*Prentice Hall*)

These two volumes bring together much of the substantial body of theory accumulated over the last decade from studies of the many models introduced to formalise various aspects of compilers.

Volume 1 commences with a review of mathematical concepts and a short chapter providing both an overview of compilers and a brief review of methods for specifying the syntax and semantics of programming languages. Chapter 2 completes the preliminaries with a thorough survey of regular sets, context free languages and the related finite and pushdown automata; this chapter might well serve as a text in a course covering these topics. Chapter 3 introduces formalisms for translation which are elaborated subsequently. Here we meet syntax directed translation schemata, finite and pushdown transducers; the relatively simple lexical analysis phase of compilation is treated in terms of regular expressions and finite transducers and then the subject of parsing is introduced. The intuitive notions of top down and bottom up parsing and their connection with left and right parses is discussed.

The considerable attention which the parsing problem has received in the literature is reflected in the fact that the next five chapters, rather less than half of the total work, are devoted to it; there is very little missing here. Chapter 4 deals with general parsing methods for context free grammars and includes the Cocke-Younger-Kasami method and that of Earley. Chapter 5, on one pass methods without backtracking, treats all of the main models which have been used in compilers, *LL*(*k*), *LR*(*k*), the many variants of precedence parsing and Floyd-Evans productions. Chapter 6 covers limited backtrack methods of both top down and bottom up varieties.

Starting Volume 2, Chapter 7 is concerned with some techniques for improving time and space requirements of various parsing methods and Chapter 8 develops the theory of deterministic parsing, establishing inclusion and equivalence relations between the language classes recognised by different deterministic parsers.

Chapter 9 returns to the subject of translation, dealing with intermediate representations of programs, models for code generation and syntax directed translation methods in the contexts of deterministic and backtracking parsing algorithms. Chapter 10 deals with the problems of storage and retrieval of semantic information for tokens, such as identifiers. In addition to the conventional solution using symbol tables and hashing functions, the theoretically interesting but (impractically?) expensive property grammars of Stearns and Lewis are examined. The final chapter presents the emerging theory underlying machine independent aspects of code optimisation. Program transformations eliminating both useless assignment statements and redundant computations are considered in the context of increasingly general environments—first within sequences of assignment statements then utilising algebraic properties, commutativity, associativity, etc. of certain operators, and finally, using flow analysis techniques, in the context of program loops. Other optimisations, code motion from within loops, reduction in strength of operators, efficient allocation of registers, all receive attention.

The presentation is formal, proofs are presented for the major theorems and lemmas but details are occasionally left and included in the many exercises at the end of chapter subsections. These exercises also serve to amplify, or to introduce additional, ideas. Bibliographical notes at the end of these subsections refer to the original papers listed in an extensive bibliography. (Volume 2 contains a composite bibliography for both volumes.) Great care has been taken in proofreading; for books of this typographical complexity errors are few in number.

This is undoubtedly a valuable reference work for those committed to improving compiler technology and for those interested in formal languages and it is very welcome. It contains a wealth of material, examples and exercises which would prove useful in a course on compilers, but one might quarrel with the authors' recommendations on its use as a textbook in such courses. A number of topics which can have far reaching effects on the overall design of a compiler are not mentioned at all; examples are runtime diagnostics, runtime storage administration and the treatment of procedures and parameters. Their omission in a work on the theory of compilers is not surprising; currently there is little of mathematical significance to say. Their omission from a course on compilers, or their relegation to a laboratory course as matters of implementation detail, is a different issue.

Similarly one might also question the value of studying so many parsers in the detail suggested. The parsing problem, notwithstanding the attention it has received, has never been large in relation to the total problems of constructing a compiler. The *LR* methods of parsing, which have not been so fully discussed elsewhere, have considerable attractions from a practical viewpoint. They will surely completely displace several of the methods recommended for detailed study. It would be quite reasonable in a course simply to present the relatively straightforward reasoning which leads to (but tends to be obscured by) the formal descriptions of the algorithms which produce parsing tables. This would allow those students who wished to do so to pursue the details of the formal descriptions more readily and it would provide sufficient background for understanding more readily and it would provide sufficient background for understanding the simple algorithms which interpret parsing tables to produce a parse. It is certainly not necessary that a compiler designer be conversant with the details of the space-time optimisations implemented in his parsing table generating program any more than it is necessary to comprehend the details of implementation in compilers to design good programs. Familiarity with general principles will help and suffice for both.

In summary, I liked these books, they contain much that is not available elsewhere. They will certainly influence my teaching but I would find it necessary both to supplement and to prune vigorously to provide a balanced view in a course on compilers.

J. Eve (Newcastle)