Reprinted from the COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY
Volume 6, Number 1, January 1963
Made in U.S.A.

R. W. BEMER, Editor

Arithmetizing Declarations: An Application to COBOL*

MELVIN E. CONWAY
L. G. Hanscom Field, Bedford, Mass.
JOSEPH SPERONI
Case Institute of Technology, Cleveland, Ohio

Introduction

The compiling technique of discovering the syntactical structure of a well-formed formula of a language by using a "syntax-directed" analysis [1], as opposed to incorporating the analysis in machine code, is equivalent in idea to performing a calculation with an interpretive system instead of coding it directly in machine language. The direct method is potentially the faster because interpretation is not required, but in many cases the interpretive method requires significantly less storage and is therefore chosen by necessity. Certain methods not available to the purist render the speed difference between the interpretive and direct methods imperceptible; with the availability of these methods the use of syntax-directed techniques becomes the only rational choice. Such techniques have been used at the Case Institute of Technology in the design of a two-pass COBOL translator whose anticipated processing speed (measured in terms of the number of average machine instructions required to translate completely one input character) is comparable to that of the fastest algebraic translators in existence.

This paper discusses a problem in the design of every COBOL processor: gleaning information from the item description entries of the Data Division. It will be seen that the solution presented here is an "interpretive" rather than a "direct" one, in the sense that the specification to the processor of predefined interrelationships among elements of the source language is expressed not in machine code but in some nonmachine language which is interpreted during compilation.

The entries of COBOL item descriptions contain clauses like "USAGE IS COMPUTATIONAL" which ascribe properties to the item in question; in this case the item is declared to occur predominantly in arithmetic calculations. The item is numeric by implication and therefore the clause named renders the clause "CLASS IS NUMERIC" superfluous. At the same time it would render the clause "CLASS IS ALPHABETIC" contradictory The goal sought here is to devise an arithmetic procedure which 1) makes clear all of the properties declared, both explicitly and implicitly, and 2) discovers any inconsistencies among these properties. Those who take seriously the application of these goals to COBOL know that the problem is not simple. There is hope that the methods presented here have more general utility in semantic analysis, e.g., resolution of meaning in mechanical translation and the formalization of the fringe area between syntax and semantics of formal languages.

* The work described here was performed when both authors were at Case Institute of Technology in 1961, and was supported in part by the UNIVAC Division of Sperry Rand Corporation.

Theory

We shall be considering vectors and matrices whose elements are 0 and 1 with the following arithmetic defined for the operators $\vee$ and $\wedge$:

$$0 = 0 \vee 0$$

$$1 = 1 \vee 1 = 1 \vee 0 = 0 \vee 1$$

$$0 = 0 \wedge 0 = 1 \wedge 0 = 0 \wedge 1$$

$$1 = 1 \wedge 1$$

That is, we are doing Boolean arithmetic.

We shall represent (column) vectors of these elements by lower case boldface letters, e.g., $\mathbf{a}$; matrices will be represented by upper case boldface letters, e.g., $\mathbf{A}$.

If a thing x admits of n properties the presence or absence of each of which it is meaningful and interesting to consider, then a *description* of x is an n-position vector $\mathbf{x}$. In particular, we say that $\mathbf{x}$ asserts that property number i is present (or that x has property i) if $\mathbf{x}_i = 1$. The case $\mathbf{x}_i = 0$ is not an assertion that property number i is absent; rather, it is no assertion at all. That is, we shall not be concerned with any assertions about the absences of properties. For example, let h be a human hand and let property 1 of h be that h is a right hand and property 2 of h be that h is a left hand. Ignoring the fact that every hand has at least one of these two properties, let us consider the truth that every hand has at most one of these two properties. That is, no hand is both a left and right hand. Therefore, there may be hands h such that

$$\mathbf{h} = \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix}$$

is a true assertion and other hands h such that

$$\mathbf{h} = \begin{pmatrix} 0 \\ 1 \\ \vdots \end{pmatrix}$$

is a true assertion, but there are no hands h such that

$$\mathbf{h} = \begin{pmatrix} 1 \\ 1 \\ \vdots \end{pmatrix}$$

is a true assertion. Properties 1 and 2 are said to be inconsistent. We can represent the consistency relationships among n properties of a thing x by an $n \times n$ *consistency matrix* $\mathbf{S}$ such that $\mathbf{S}_{i,j} = 0$ if properties i and j are consistent, and $\mathbf{S}_{i,j} = 1$ if properties i and j are inconsistent. Clearly, for every x with consistency matrix $\mathbf{S}$, $\mathbf{S}$ is a symmetric matrix with a zero diagonal. Let $\mathbf{H}$ be the consistency matrix of h in our example. Then

$$\mathbf{H} = \begin{pmatrix} 0 & 1 & \cdots \\ 1 & 0 & \\ \vdots & & \end{pmatrix}$$

If $\mathbf{a}$ is a description of x, we say that $\mathbf{a}$ is an inconsistent description of x iff[1] it asserts the existence of inconsistent properties. $\mathbf{a}$ is consistent iff it is not inconsistent. For example, if properties 3 and 7 of x are inconsistent, then $\mathbf{a}$ is inconsistent if $\mathbf{a}_3 = 1$ and $\mathbf{a}_7 = 1$.

THEOREM 1. *Let* $\mathbf{a}$ *be an n-position description of* x *Let* $\mathbf{S}$ *be the* $n \times n$ *consistency matrix of* x. $\mathbf{a}$ *is inconsistent iff* $\mathbf{a}^T\mathbf{S}\mathbf{a} = 1$.

PROOF. The matrix operations are defined by drawing the analogy between $\vee$ and $+$, and between $\wedge$ and $\times$. $\mathbf{a}$ is inconsistent iff

$$(\exists i)\,(\exists j)\,(\mathbf{S}_{i,j} = 1 \quad \text{and} \quad \mathbf{a}_i = 1 \quad \text{and} \quad \mathbf{a}_j = 1)$$

iff

$$(\exists i)\,(\exists j)\,(\mathbf{a}_i \wedge \mathbf{S}_{i,j} \wedge \mathbf{a}_j = 1)$$

iff

$$\bigvee_{i=1}^{n} \bigvee_{j=1}^{n} (\mathbf{a}_i \wedge \mathbf{S}_{i,j} \wedge \mathbf{a}_j) = 1.$$

But

$$\bigvee_{i=1}^{n} \bigvee_{j=1}^{n} (\mathbf{a}_i \wedge \mathbf{S}_{i,j} \wedge \mathbf{a}_j) = \mathbf{a}^T\mathbf{S}\mathbf{a}.$$

The step from the use of $(\exists i)$ to the use of $\bigvee_{i=1}^{n}$ is based on two lemmas:

1. $(\exists i)\,(a_i = 1)$ *iff* $a_1 = 1$ *or* $a_2 = 1$ *or* $\cdots$ *or* $a_n = 1$.
2. $(a = 1$ *or* $b = 1)$ *iff* $(a \vee b = 1)$.

Property i is said to be a consequence of property j if i occurs whenever j does, that is, whenever the existence of j implies the existence of i. Consequencehood is a transitive relation.

$\mathbf{a}$ is a *partial description* of x when the existence of every property possessed by x is either asserted by $\mathbf{a}$ or is implied by some property whose existence is asserted by $\mathbf{a}$, and when $\mathbf{a}$ does not assert the presence of a property not possessed by x.

$\mathbf{b}$ is a *complete* description of x when $\mathbf{b}$ is a partial description of x which asserts the existence of every property possessed by x. A partial and complete description of x both imply everything about the properties of x, but the complete description is explicit about every property. In this sense, a complete description does make assertions about the absence of properties, namely, those whose corresponding elements of the description are zero.

The specification of consequencehood among the properties of x is expressed in a *contingency matrix* $\mathbf{T}$ of x such that $\mathbf{T}_{i,j} = 1$ iff property i is a consequence of property j. $\mathbf{T}$ is not in general symmetric, but $\mathbf{T}_{i,i} = 1$.

THEOREM 2. *Let* $\mathbf{a}$ *be a partial description of* x, *let* $\mathbf{b}$ *be a complete description of* x, *and let* $\mathbf{T}$ *be the contingency matrix of* x. *Then* $\mathbf{b} = \mathbf{T}\mathbf{a}$.

PROOF. Consider the ith element of either side. $\mathbf{b}_i = 1$ iff property i is present iff $\mathbf{a}_i = 1$ or $(\exists j)\,(\mathbf{T}_{i,j} = 1$ and $\mathbf{a}_j = 1)$. Now notice that $\mathbf{a}_i = 1$ implies $(\exists j)\,(\mathbf{T}_{i,j} = 1$ and $\mathbf{a}_j = 1)$, because $\mathbf{T}_{i,i} = 1$. Therefore $\mathbf{b}_i = 1$ iff $(\exists j)\,(\mathbf{T}_{i,j} = 1$ AND $\mathbf{a}_j = 1)$ iff $(\exists j)\,(\mathbf{T}_{i,j} \wedge \mathbf{a}_j = 1)$ iff $\bigvee_{j=1}^{n} \mathbf{T}_{i,j} \wedge \mathbf{a}_j = 1$. Thus

$$\mathbf{b}_i = \bigvee_{j=1}^{n} \mathbf{T}_{i,j} \wedge \mathbf{a}_j .$$

That is, $\mathbf{b} = \mathbf{T}\mathbf{a}$.

Going back to the example of h, let property 3 be that of being in the act of holding the bearer's left hand. Then the contingency matrix $\mathbf{T}$ of h looks like this:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 1 & \cdots \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \\ \vdots & & & \end{pmatrix}.$$

Consider now a set X of things x, each with the same set of n properties about which it is meaningful and interesting to inquire. For example, consider the set H of twelve things consisting of a person's two hands and their ten associated fingers. Each element of H has its own $\mathbf{S}$ and $\mathbf{T}$ matrices which impose a structure on its properties. Additionally, there are meaningful consistency and contingency relations which exist between ordered pairs of elements of H. Let property number 4 be that of being fully immersed in water, and let property number 5 be that of having five fingers. Property 4 of either hand implies property 4 of any of its fingers, but rarely can the same be said of property 5.

Thus by analogy with the definitions above we may define a *cross-consistency matrix* $\mathbf{P}$ *between* x *and* y such that $\mathbf{P}_{i,j} = 1$ iff it is inconsistent for x to have property i and

[1] If and only if.

for y to have property j. Notice that $\mathbf{P}$ is not in general symmetric. If $\mathbf{a}$ is a description of x and $\mathbf{b}$ is a description of y, then $\mathbf{a}$ and $\mathbf{b}$ are said to be inconsistent if for some i and j, $\mathbf{a}_i = 1$ and $\mathbf{b}_j = 1$ and $\mathbf{P}_{i,j} = 1$. Thus, using this notation, we have

THEOREM 3. $\mathbf{a}$ *and* $\mathbf{b}$ *are inconsistent iff* $\mathbf{a}^T\mathbf{P}\mathbf{b} = 1$.

Also by analogy we define a *cross-contingency matrix* $\mathbf{Q}$ *between* x *and* y as follows: $\mathbf{Q}_{i,j} = 1$ iff property i of x is a consequence of property j of y.

Application to COBOL

Item description entries of the COBOL Data Division occur in a linear list but are implicitly structured, by virtue of accompanying level numbers, as nodes of trees. In the language of the above theory, each item description is equivalent to a "partial description" of a "thing" (an item) with the following[2] 17 properties:

1. Display usage
2. Computational-1 usage (one-word fixed point)
3. Computational-2 usage (two-digit representation of numerics)
4. Computational-3 usage (floating point)
5. Numeric class
6. Alphabetic class
7. Alphanumeric class
8. Signed
9. Sign is data-name
10. Point location declared
11. Editing (except for zero insertion)
12. Justified
13. Synchronized
14. Occurs integer times
15. Value is numeric literal
16. Value is non-numeric literal
17. Use of data-name in procedure division requires subscript

Size is a separate specification not conveniently expressible in these terms. Information in a PICTURE may be reduced to some subset of these properties and size. When the compiler reaches the period at the end of the item description entry it can create a 17-position description vector which states what declaration clauses (each corresponding to a property) were present in that entry. This vector has the characteristics discussed in the theory section above. The compiler table entry which must be created from the description entry has many fewer than 17 items; class and usage, for example, may be expressed in a single digit. This redundancy of the language suggests that there is a contingency matrix describing certain implicit relations between the clauses, and indeed there is. At the same time certain clauses are mutually exclusive, for example, SIGNED and SIGN IS data-name. These relations may be described with a consistency matrix. Since these two matrices cannot both contain a 1 in the same position, both are given in Figure 1 in a single array in order to save space here; 1 represents 1 in the contingency matrix; x represents 1 in the consistency matrix; other positions contain 0.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	x	x	x													
2	x	1	x	x		x	x		x		x					x	
3	x	x	1	x		x	x		1		x					x	
4	x	x	x	1		x	x		x	x	x	x				x	
5		1	1	1	1	x	x	1	1	1	x					x	
6		x	x	x	x	1	x	x	x	x	x				x		
7		x	x	x	x	x	1	x	x	x	1				x		
8						x	x	1	x		x					x	
9		x		x		x	x	x	1		x					x	
10				x		x	x			1	x	x				x	
11		x	x	x	x	x		x	x	x	1				x		
12				x						x		1					
13													1				
14														1	x	x	
15						x	x				x			x	1	x	
16		x	x	x	x			x	x	x				x	x	1	
17														1			1

FIG. 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	x	x	x				x	x	x	x						
2	x	1	x	x		x	x	x	x	x	x					x	
3	x	x	1	x		x	x	x	x	x	x					x	
4	x	x	x	1		x	x	x	x	x	x				x	x	
5		1	1	1	1	x		x	x	x	x				1		
6		x	x	x	x	1		x	x	x	x				x		
7		x	x	x	x	x		x	x	x	x				x		
8						x		x	x	x	x						
9		x		x		x		x	x	x	x						
10				x		x		x	x	x	x						
11		x	x	x	x	x		x	x	x	x				x		
12								x	x	x	x						
13								x	x	x	x						
14								x	x	x	x						
15						x		x	x	x	x			x	x	x	
16		x	x	x	x			x	x	x	x			x	x	x	
17								x	x	x	x			1			1

FIG. 2

So much for single item description entries. A hierachy of item description entries describes a set of things in the same sense as discussed in the theory. Interactions (i.e., nonzero cross-consistency and cross-contingency matrices) do not occur between all ordered pairs of entries, but only between immediate neighbors on a tree. More exactly, nonzero cross-consistency and cross-contingency matrices between x and y occur iff x has a level number greater than that of y, x follows y in the Data Division, and no entry occurs between y and x with level number less than or equal to that of y; that is, x is immediately below y in the hierarchy. The two matrices are expressed in Figure 2 in one array as before.

[2] These properties are conceived for an internally decimal-word machine like the Burroughs 220, IBM 7070 and UNIVAC Solid-State 80.

Notice that certain columns of the cross-consistency matrix contain all 1's (x's). This expresses the fact that the corresponding properties must be declared for elementary items only: if an item with any property at all is declared at a lower level an inconsistency will result. Of course an item can be declared with no properties except size, and then the inconsistency would not be caught. This difficulty can be avoided by adding an 18th property: that of merely existing; every description vector would have a 1 in the 18th position. Then the device of specifying properties which can occur only at the elementary item level by writing a column of ones in the cross-consistency matrix will work; in fact a single 1 in the 18th row will do.

The method employed by the compiler is this. As each entry is scanned a description vector is produced from the clauses present. A stack of description vectors (maximum size = 49) is maintained so that at all times the vector of the immediate superitem is available. Let $\mathbf{x}$ be the description vector obtained by scanning the current entry (with 1's corresponding to the clauses present), and let $\mathbf{w}$ be the description vector of the immediate super item. Let $\mathbf{S}$, $\mathbf{T}$, $\mathbf{P}$ and $\mathbf{Q}$ be the consistency, contingency, cross-consistency and cross-contingency matrices given above; they are the same for all pairs $\mathbf{x}$ and $\mathbf{w}$.

First fill out $\mathbf{x}$ by adding to it the properties inherited from the superitem. This defines $\mathbf{y} = \mathbf{Q}\mathbf{w} + \mathbf{x}$. Now, since $\mathbf{y}$ may omit some implicit declarations, expand $\mathbf{y}$ by "developing" all the implied properties: $\mathbf{z} = \mathbf{T}\mathbf{y}$. Now $\mathbf{z}$ is the description vector for the current item; $\mathbf{z}$ is what is placed on the stack if it becomes necessary to descend to a lower level (i.e., if the next entry has a greater level number less than 50 or equal to 88). $\mathbf{z}$ is checked for consistency with itself: $a = \mathbf{z}^T\mathbf{S}\mathbf{z}$, and for consistency with the declarations of the higher-level items: $b = \mathbf{z}^T\mathbf{P}\mathbf{w}$. The choice of clauses in the current item description entry is consistent iff $a = b = 0$. It remains for the program which constructs table entries from the parameters supplied to decide whether the information is sufficient. (Usually size and class clauses are sufficient.)

Certain conditions, such as the desirability of redundant clauses, have not been discussed. For example, the COBOL manual might be interpreted as insisting that a VALUE IS numeric-literal clause be accompanied by a CLASS IS NUMERIC clause (or equivalent) at the same or higher level. Such checks are not available to the method presented here. One might consider, in the case cited here, adding a 19th property, that of **not** having been declared numeric, and checking for consistency with the value clause. But there is no way to turn a 1 in the 19th position into a zero if the item inherits a numeric declaration from a superitem.

Realization on a Computer

The matrix operations discussed above are readily implemented on a machine with an AND ($\wedge$) or OR ($\vee$) instruction. The present discussion applies equally well to binary or decimal machines, providing only the digits 0 or 1 occur. An n-vector is represented as a sequence of n bits within a nonnegative word. If the word size is too small the matrices may be partitioned.

First, a vector may be complemented by subtracting it from a positive word of all ones. The complement of $\mathbf{x}$, in which ones and zeros are reversed, is written $\neg\mathbf{x}$. Addition of vectors is realized as follows. If the machine has an OR instruction, then $\mathbf{a} \vee \mathbf{b}$ is $\mathbf{a}$ OR $\mathbf{b}$. If the machine has an AND instruction, an application of DeMorgan's identity suffices: $\mathbf{a} \vee \mathbf{b}$ is $\neg(\neg\mathbf{a}$ AND $\neg\mathbf{b})$. If it is necessary to test whether $\mathbf{a} \vee \mathbf{b}$ is zero it is sufficient to add the two words numerically and test the sum for zero.

Let $\mathbf{a}^T \wedge \mathbf{b}$ be the scalar product of $\mathbf{a}$ and $\mathbf{b}$, and let $(\mathbf{a}, \mathbf{b})$ be the vector each of whose elements is the product of the two corresponding elements of $\mathbf{a}$ and $\mathbf{b}$. $(\mathbf{a}, \mathbf{b})$ is $\mathbf{a}$ AND $\mathbf{b}$. Notice that $(\mathbf{a}, \mathbf{b}) = \mathbf{0}$ if and only if $\mathbf{a}^T \wedge \mathbf{b} = 0$. This result eliminates the necessity to calculate $\mathbf{a}^T \wedge \mathbf{b}$ directly by summing products of elements.

A matrix $\mathbf{A}$ is represented as a set of row vectors $\mathbf{A}_{i,*}$. $\mathbf{A}\mathbf{x}$ is computed element by element, the ith element being zero if $\mathbf{A}_{i,*} \wedge \mathbf{x}[= (\mathbf{A}_{i,*}, \mathbf{x})] = 0$; otherwise it is 1. Once this matrix-vector multiplication is accomplished the bilinear form $\mathbf{y}^T\mathbf{A}\mathbf{x}$ is simple: $\mathbf{y}^T\mathbf{A}\mathbf{x} = \mathbf{y}^T \wedge (\mathbf{A}\mathbf{x}) = \mathbf{y}^T \wedge \mathbf{z}$, letting $\mathbf{z} = \mathbf{A}\mathbf{x}$. Then $\mathbf{y}^T\mathbf{A}\mathbf{x} = 0$ iff $(\mathbf{y}, \mathbf{z}) = \mathbf{0}$.

Conclusion

Using a 10-digit decimal machine (in this case the Burroughs 220), two words are required for each vector; approximately 250 words are required for the stack and four matrices. A binary machine with a respectable word size requires at most half as many words. In this application the "interpretive" method is clearly faster than the "direct" method coded to perform as thorough a check, and may even require less space. Furthermore, there is practically nothing to debug, a fact not to be ignored by COBOL compiler writers.

REFERENCE

1. IRONS, E. T. A syntax directed compiler for ALGOL 60. *Comm. ACM 4*, 1 (Jan. 1961).