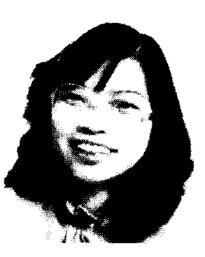
Methodology for Security Analysis of Data-Processing Systems

John M. Carroll * and Oi-Lun Wu
Computer Science Department, University of Western Ontario, London, Ontario N6A 5B9, Canada

Use of the logic programming language PROLOG assists in tracking functional relationships through complex networks to ensure that all products receive a security classification consistent with the operative security model. The language also permits computing work factors for data products. In some cases, these work factors may modify the levels of protection required.

Keywords: Data-flow diagrams, Data-flow models, Artificial intelligence, PROLOG Work factor, Relational algebra, Relational data bases, Classification, Cost/benefit analysis, "Star"-property, Security models

John M. Carroll is a Professor in the Computer Science Department of the University of Western Ontario, London, Canada. During 1975–1976, Professor Carroll was consultant to the Electronic Data Processing Security section of the Royal Canadian Mounted Police, where he prepared draft standards on computer security for the federal government of Canada. During 1970–1972, he was consultant to the federal Privacy and Computers Task Force. His 1969–1970 study on the privacy and security of university student records, prepared under the patronage of the Canadian Council, was one of the earliest Canadian efforts in this field. Dr. Carroll is the author of *Computer Security* and *Confidential Information Sources: Public and Private*. His research interests include automatic analysis of documents, computer-aided risk analysis of computer systems, computer simulation of municipal protective services, and data protection for microcomputers.



Oi-Lun Wu is currently enrolled as a computer science graduate student at the University of Western Ontario. She received a Certificate of Honours Standing in 1982 and a B.Sc. from Brock University in 1981. She majored in computer science as an undergraduate and was a programmer and programming consultant with the Social Science Computing Laboratory at Western Ontario. Prior to that, she was a tutor at Brock.

* Presently with Computer Science and Systems Branch, Naval Research Laboratory, Washington, DC 20375, USA.

1. Introduction

Rigorous application of the principle of preserving the " *-property" in a data-processing system suggests that if such a system includes a sensitive data item, any other data item co-mingled with it or derived from it, and any process used to realize such co-mingling or derivation should attract at least as high a security classification as the data item in question.

Tracking functional relationships through a complex data-processing system can be difficult; it is easy to make errors or omissions, or to apply assumptions inconsistently. We believe that use of a logic programming language can help circumvent these difficulties.

Even when data-flow models are applied correctly, they upset some users by classifying many data items and processes that the users do not believe deserve protection. In cases involving national security, this is precisely what they should do.

In the private sector, however, the user is free to decide whether the data security dictated by a model is cost/effective or not. One way to do this is to associate a work factor with every data item and then estimate whether the cost to an adversary of exploiting perceived security exposures is greater than the benefit to him of the information he can derive thereby. If so, one can assert that work factor security shall have been achieved. The computational capability of the logic programming language PROLOG allows making such work factor estimates.

2. Preliminary Analysis: Data-Flow Diagrams

The first step in analyzing a data-processing system is to describe it in terms of a data-flow diagram. Fig. 1 is a conventional logic flow chart depicting dataprocessing operations in a merchandising establishment that sells both for cash and credit. Fig. 2 is a dataflow diagram of the system. The diagram conforms to a standard

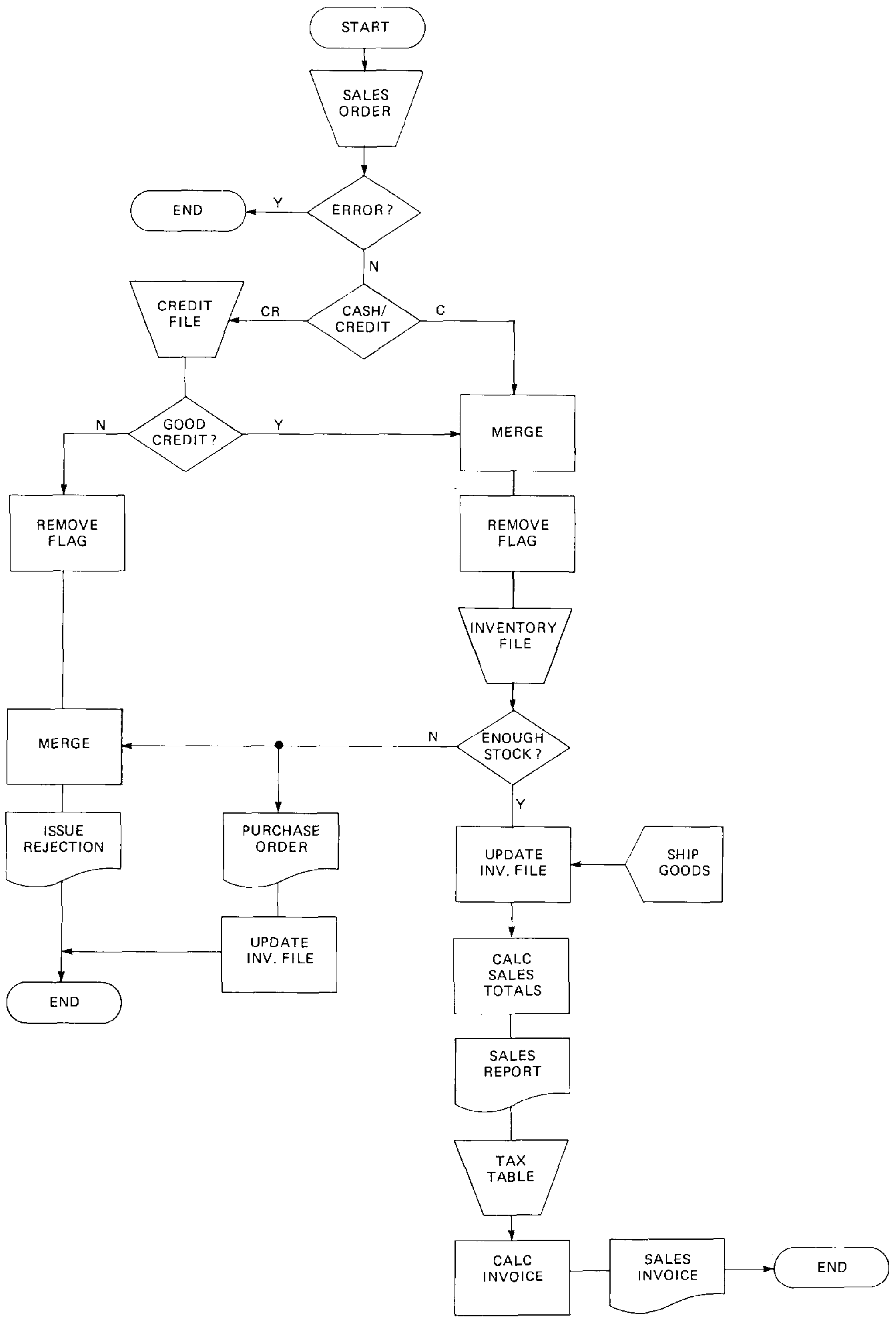


Figure 1. Conventional Logic Flow Chart of Order-Fulfillment System.

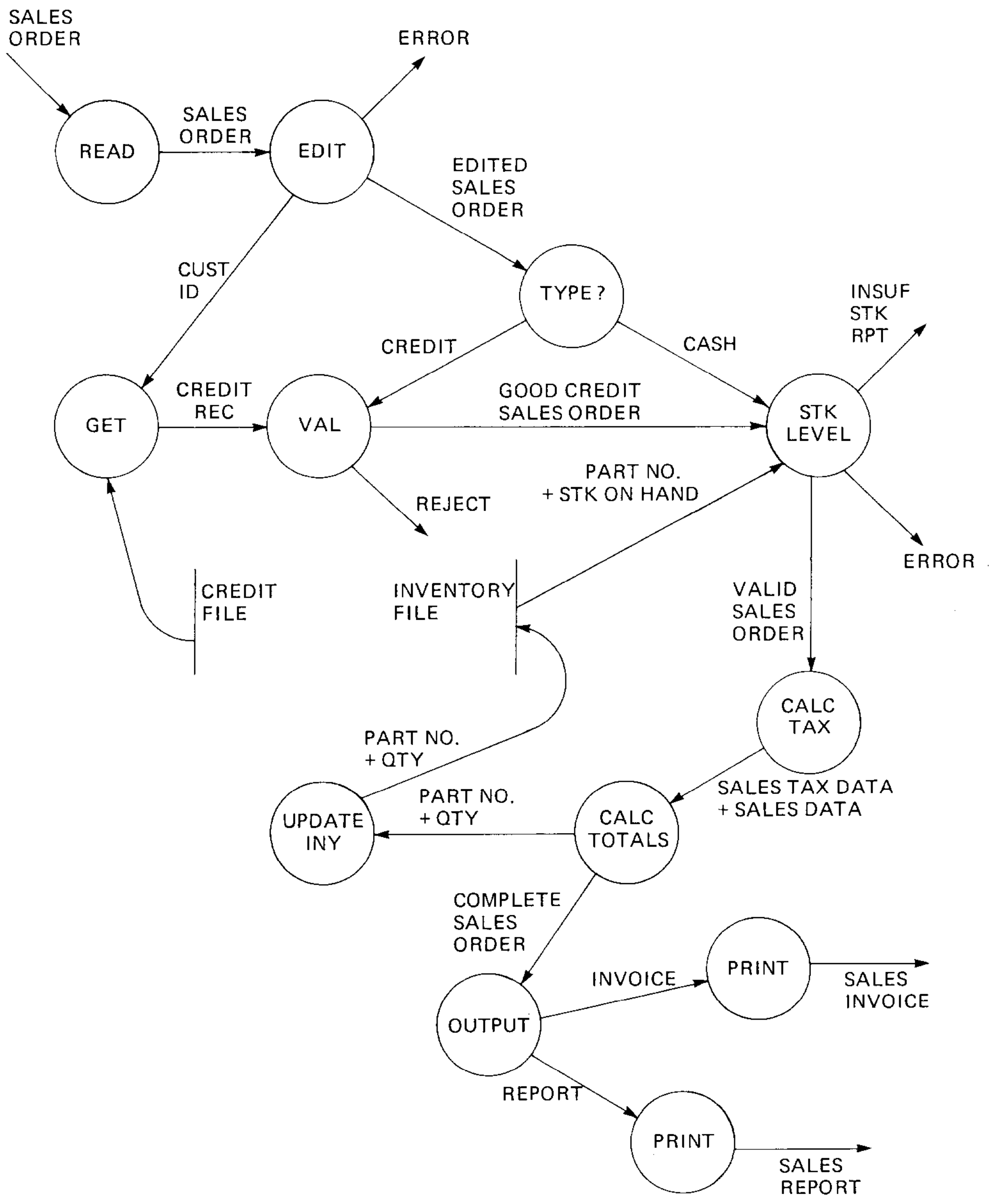


Figure 2. Data-Flow Diagram of Order-Fulfillment System.

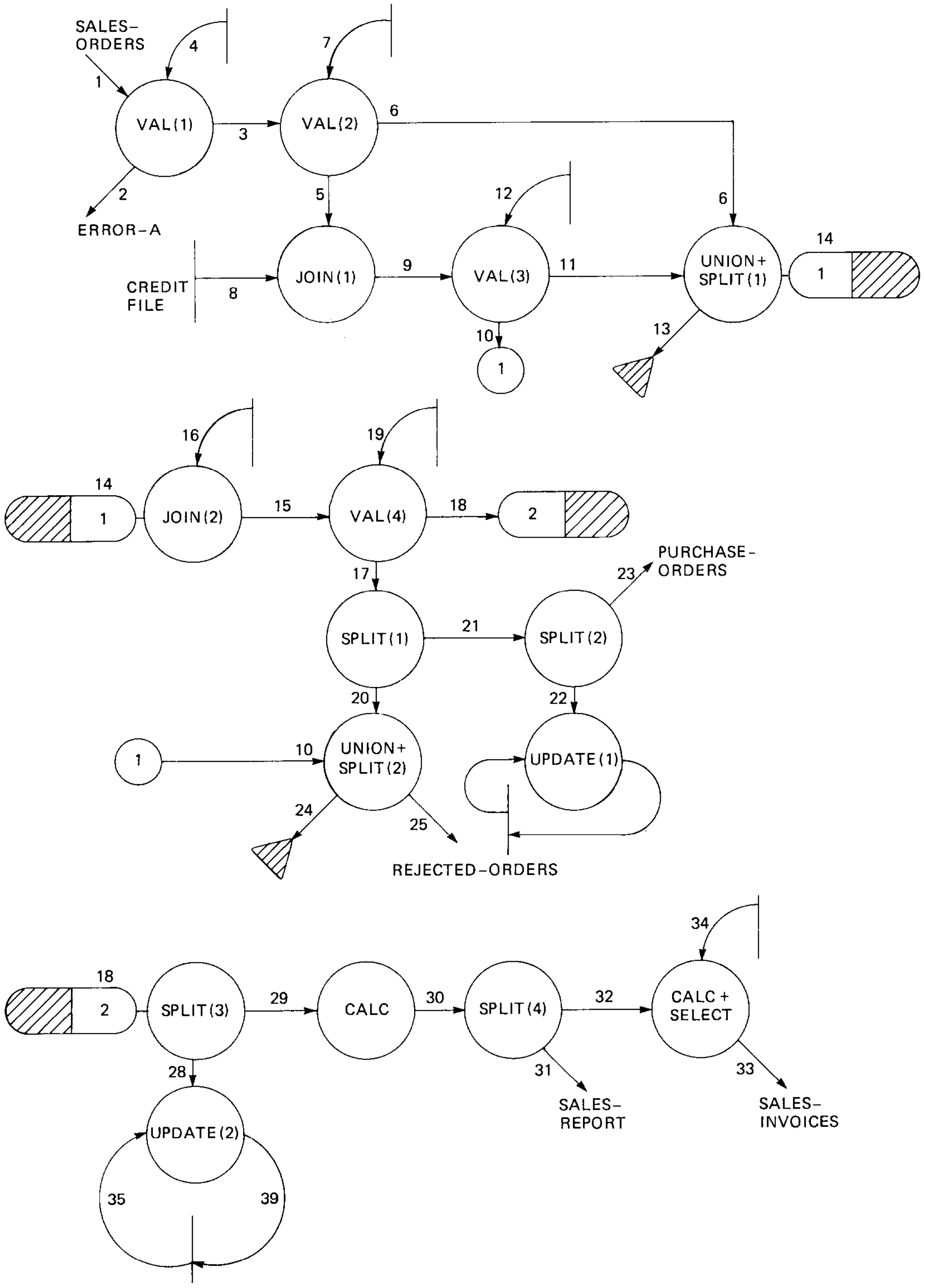


Figure 3. Modified Data-Flow Diagram.

promulgated by the Canadian Department of National Defence. A circle or "bubble" denotes an operation. A vertical bar denotes a file, table, or internally stored value. A directed arc, that is, a line with an arrowhead, denotes a data item entering or leaving an operation.

We have found the following conventions useful in preparing a data-flow diagram for analysis:

1. No operation may have more than two inputs or two outputs.

2. The arcs will enter or leave bubbles according to the following schedule of bearings:

045	output to another data processing system
090	output to another operation
135	output
180	output to another operation
225	output of errors or exceptions
270	input from another operation
315	input
360	input from another operation

3. Operations will be expanded in terms of relational algebra primitives and macros:

JOIN	dyadic input; monadic output
PROJECT	monadic input; monadic output
SELECT	dyadic input; monadic output (one input is an exemplar)
INTERSECT	dyadic input; monadic output
SUBTRACT	dyadic input; monadic output
UNION	dyadic input; monadic output
CALCULATE	monadic or dyadic input; monadic output
SPLIT	monadic input; dyadic output (macro of PROJECT and COPY)
VALIDATE	dyadic input; dyadic output (macro of SELECT and COPY)
UPDATE	dyadic input; monadic output (one input may be a file) (macro of UNION and SUBTRACT)

Linear Combinations – such as: UNION + SPLIT; CALCULATE + SPLIT

4. The "start" symbol denotes data-flow continuation. The symbols appear in pairs. Each symbol is bifurcated and serially numbered. The right-hand half of the input member and the left-hand half of the output member are shaded.

5. The "on-page connector" symbol is used to avoid arc crossovers.

6. An arc terminating in an inverted triangle indicates that the data item is to be stored in a file. If the triangle is shaded; that file is to be destroyed.

7. Input arcs, output arcs, output-to-other-systems arcs, output-of-errors arcs, and vertical bars denoting files can be drawn with double weight.

Fig. 3 is a version of Figure 2 redrawn to reflect these conventions.

The external input is "sales-orders"; it is validated against "template = A"; "valid-orders" proceed, "error-A" are intercepted for correction and reentry.

The "valid-orders" are tested against "credit/cash-criteria"; "credit-sales" are joined with a "credit-file" to produce "credit-sales + ratings". These are tested against "credit-criteria"; "good-credit-sales" are united with "cash-sales" to produce a file called "good-sales". The flag that indicates whether a sale is cash or credit is stripped from this file.

The "good-sales" are joined with the "inventory-file" to produce "sales + quantity-on/hand" which are tested against "shipping-criteria" to produce "fillable-sales" and reject-sales-shortage". The latter are split into "reject-sales-shortage-1" and "shortage". Information that would identify individual customers is deleted from "shortage". The "shortage" file is then used to prepare "purchase-orders" and to update the "inventory-file", which is now called "inventory-file + orders".

The "reject-sales-shortage" file is merged with the "reject-sales-credit" file and distinguishing flags (that is, credit vs cash) are stripped off. The resulting file is called "rejected-orders"; "rejected-orders" are returned to the customer.

The "fillable-sales" are split into "issues," which are used to update the "inventory-file + orders-1" file to "inventory-file + orders + issues"; and "sales", which are extended and sub-totaled to produce "sales + totals".

The "sales + totals" file is split into "sales-report," from which customer identification is removed; and "sales + totals-1." The latter are run against the "sales-tax-table" to produce "sales-invoices" that are sent to the customer.

3. Data-Processing Security Model

All data files, tables or items are regarded as being relations. A relation of degree M and cardinality N, has M attributes and N records. A sensitive relation is defined as one which has one or more sensitive attributes and one or more sensitive records.

Conservative security models dictate that a relation that is functionally related to a sensitive relation must attract a security classification at least as high as that of the sensitive relation. Policies like this frequently lead to what some people regard as overclassification. This, in turn, may give rise to costs that users, especially those in the private sector, see as unacceptable. On the other hand, normal processing may actually decrease the sensitivity of sensitive relations. A functional relationship between relations may result in an output relation that is less sensitive than the more sensitive input relation. There are at least five ways this can occur:

1. Randomly mix sensitive and nonsensitive records.
2. Randomly select from a mix of sensitive and nonsentive records.
3. Remove sensitive attributes.
4. Calculate using an irreversible function (such as bit-wise AND), a randomly selected function, or an unknown function; or calculate using an unknown variable or a random variate. An example of this tactic would be encryption.
5. Select only nonsensitive records.

Assume a relation contains S sensitive records and N nonsensitive records randomly mixed and indistinguishable from each other. We assert that the work factor, a measure of the work required to obtain sensitive information from that relation, is

$$W = (N/(N + S)) * 10$$

Suppose we produce the union of two Boolean compatible and independent relations R1 and R2; the resulting work factor would be

$$W1, 2 = ((N1 + N2)/(N1 + N2 + S1 + S2)) * 10$$

If relations R1 and R2 were completely dependent and R1 had more records, then the work factor would be

$$W1, 2 = (N1/(N1 + S1)) * 10 \text{ or simply } W1$$

If the outcome of R1 UNION R2 is unknown, we can estimate the result by computing the mean of W1, 2 and W1.

The work factor obtained by using a secure encryption system might be 10. The work factor attached to a sensitive and unprotected relation is 0. Splitting off sensitive attributes might earn a work factor of 8 or 9. Table 1 gives an algorithm for computing the work factor for each of the operation types shown in Fig. 3.

4. Security Analysis

In our example, "credit-file" is regarded as the sensitive relation. We wish to perform a security analysis to determine two things:

1. What relations are functionally related to the sensitive relation and therefore deserving of protection?
2. What are the work factors of these relations?

It is important that the analysis be complete,

Table 1
Work-Factor Computations

Operation	Maximum	Minimum	Expected
Union	(N1+N2)/(T1+T2)	N1/T1	(2N1+N2)/(2T1+T2)
Intersect	N2/T2	undefined	N2/T2
Subtract	N1/T1	(N1−N2)/(T1−T2)	(2N1−N2)/(2T1−T2)
Join	N2/T2	undefined	N2/T2
Select	10	N1/T1	2N1/(N1+T1)
Project	10	N1/T1	(N1+T1)/2T1

(1) T1 = S1 + N1
(2) T2 = S2 + N2
(3) T1 > T2 = > N1 > N2
(4) N is uniformly distributed in T
(5) Select and Project operations are intended to remove sensitive records and attributes
(6) T2 is 0 for Select and Project
(7) WF is work-factor; low WF requires high security (range: 0...10)
(8) NA means relation not functionally dependent on critical relation
(9) *credit-file* is the critical relation

that rules and assumptions be applied consistently, and that the rules, assumptions, and network relationships be explicit and conveniently available for review. As larger and more complex systems become subjects for security analysis, clearly systems analysts will have to seek computer assistance. We have found the language PROLOG particularly useful in this regard.

The language was designed for logic programming and has been used in artificial-intelligence studies. It is based upon predicate calculus and makes extensive use of tree-descent and backtracking.

5. PROLOG Program for Security Analysis

We regard the program as consisting of five main parts:

1. Data bank explicating all network relationships.
2. Data bank describing all operations.
3. File data bank.
4. "Data bank" of algorithms for determining work factors.
5. Recursive driver program.
6. Question, which is the command to execute the program.

We will now describe each of these components.

5.1. Network Data Bank

Entries in this data bank assume the format:

```
next (X, Y)
```

This entry asserts that relation Y is a direct descendent (that is, son) of relation X. Each data bank entry describes only one parent and only one descendent. For example, consider the operation in which the relation "credit-sales + ratings" is validated against "credit-criteria" to produce two outputs: "good-credit-sales" and "reject-sales-credit". This operation dictates that we generate four network data bank entries:

```
next(credit-sales + ratings, good-credit-sales).
next(credit-sales + ratings, reject-sales-credit).
next(credit-criteria, good-sales-credit).
next(credit-criteria, reject-sales-credit).
```

Every input and every output must have a unique name. If the same relation participates in two or more sequential operations different appearances can be distinguished by appended sequential numerals:

```
sales + totals
sales + totals-1
```

The network data bank for the system shown in Figure 3 contains 36 entries.

5.2. Operation Data Bank

Entries in the operation data bank have the format:

```
opr(Y, O).
```

where O is an operation and Y is an output relation (that is, descendent). There must be an entry corresponding to every descendent. Consider the VALIDATE operation wherein "valid-orders" are tested against the "credit/cash-criteria" to produce two outputs: "cash-sales" and "credit-sales". In this case we generate two entries:

```
opr(cash-sales, validate).
opr(credit-sales, validate).
```

Moreover, if a different algorithm is to be used to compute the work factor of each output relation, then the operation must be given a unique name in each case. For example, when we SPLIT "reject-sales-shortage" into "reject-sales-shortage-1" and "shortage", the latter output relation consists only of attributes needed to prepare a purchase order and update the inventory file. The relation has thus been sanitized as far as communicating customer-credit information and is awarded a work factor of 9. The operation data bank entries reflect this condition:

```
opr(reject-sales-shortage-1, split).
opr(shortage, split-9).
```

To demonstrate further the dependence of operation name on work factor algorithm, consider the operation wherein "sales + totals" is SPLIT into "sales + totals-1", which retains attributes communicating customer-credit information, and "sales-report", which contains only aggregated data. The entries are:

```
opr(sales + totals-1, split).
opr(sales-report, split-9).
```

Even though the relations are different, we shall use the same work factor algorithms and for this reason, the operation names are the same.

5.3. File Data Bank

The file data bank has the format:

```
file(X, Total, Nonsensitive).
```

where X is an input relation or parent; T is equal to N + S, the total number of records; and N is the number of nonsensitive records.

The user must generate an entry in the file data bank for the initial data input (that is, "sales-orders") and for all inputs interrupting the main flow of data: "credit/cash-criteria", "credit-file", "credit-criteria", "inventory-file", "shipping-criteria", and "sales-tax-tables." In other words, for the exogenous inputs. Representative entries are:

```
file(credit-file, 1000, 0).
file(credit-criteria, 100, 100).
file(inventory-file, 1000, 1000).
```

It will be seen that the program generates file data bank entries for the internally produced (that is, endogenous) inputs.

5.4. Work Factor Data Bank

The work factor data bank is really a collection of program statements. There is in the work factor data bank one statement for each unique operation name. It defines a work factor for every uniquely named operation (for example, SPLIT, SPLIT-9, etc.).

The left side of each statement establishes our goal. It is expressed as:

```
workf(unique-operation-name, parent,
descendent, total-number-of-records, number-of-
nonsensitive-records, work-factor)
```

The first three qualifiers in the argument identify the relation for which we are computing a work factor. It is the descendent. The last three qualifiers state the quantities we shall compute for that relation. Some representative goals are:

```
workf(split-9, X, Y, Total, Nonsensitive, Workf):-
workf(split-8, X, Y, Total, Nonsensitive, Workf):-
workf(union, X, Y, Total, Nonsensitive, Workf):-
```

The connecting symbol between the left and right hand sides of the equation is read "if."

On the left hand side of the equation, we identify the main parent X and the other parent (if any) and call the file data bank for information regarding them; evaluate Total, Nonsensitive, and Workf; and store information regarding descendent Y in the file data bank. The PROLOG functions (right hand sides of equations) corresponding to the three goals given above are:

```
file(X, T1, N1), Total is T1, Workf is 9,
   Nonsensitive is 9 * Total/10,
   assert(file (Y, Total, Nonsensitive)).
file(X, T1, N1), Total is T1, Workf is 8,
   Nonsensitive is 8 * Total/10,
   assert(file (Y, Total, Nonsensitive)).
next(X, Y), next(F2, Y), X [not] = = F2,
   file(X, T1, N1), file(F2, T2, N2),
   Total is T1 + T2, Nonsensitive is N1 + N2,
   Workf is 10 * Nonsensitive/Total,
   assert(file(Y, Total, Nonsensitive)).
```

The first two cases concern one descendent of a SPLIT. There is but one parent, X; Total is evaluated directly from the file data bank entry appropriate to X. Judgment or prior knowledge permits us to set Workf at 9 and 8 respectively. Then we compute the number of nonsensitive records as equal to Workf * Total/10. Finally, we store this information regarding Y in the file data bank.

In the third case, we must identify two parents inasmuch as it concerns the UNION of two relations. We first establish the main parent (main in the sense that it is encountered first in the main flow of data). We consult the network data bank to do this. Then we establish the other parent, which we call F2. We specify that X is not equal to ([not] = = , where [not] denotes "backslash") F2. We next call the file data bank and obtain the Total and Nonsensitive values for X (T1 and N1) and F2 (T2 and N2) respectively.

Hypothesizing Boolean independence between X and F2, we compute Total of Y as equal to T1 + T2; and Nonsensitive of Y as N1 + N2. Workf of Y is computed as 10 * Nonsensitive/Total. Finally, we store this information regarding Y in the file data bank.

5.5. Driver Program

The recursive driver program consists of two statements:

```
descendent(X, Y, Workf, Total, Nonsensitive, D, O)
  :-next(X, Y), opr(Y, O), workf(O, X, Y,
  Total, Nonsensitive, Workf), depth (D).
descendent(X, Z, Workf, Total, Nonsensitive, (D, O)
  :-next(X, Y), descendent (Y, Z, Workf, Total,
  Nonsensitive, D, O).
```

The first statement establishes that Y is the son of X having a certain work factor, total number of records, number of nonsensitive records, work factor, depth of descent into the network (depth() is a PROLOG system function), and is the product of a certain data-processing operation "if" Y immediately follows X in the network, is the product of a certain data-processing operation, has a work factor that can be computed from an algorithm specific to that operation, and exists at a certain depth into the network.

The second statement establishes that Z is a descendent of X possessing the same set of qualities "if" Y is the son of X and Z is a descendent of Y.

Driven by this program, PROLOG starts where it is instructed to start; and computes a file data bank entry for every output in the main flow of data. When it reaches the end, it backtracks and follows any and all branches until it has traversed the entire network.

5.6. Question

The question tells PROLOG where to start. Its appearance in this program is:

```
:?-descendent(credit-file, X, Workf, Total,
Nonsensitive, D, O).
```

The colon (:) is the system prompt; "credit-file" is the starting value for X; the remainder of the argument presents the desired format for results. In principle, one could select several starting points corresponding to different sensitive relations within a data-processing network and perform a security analysis with respect to each one simply by repeatedly running this program. The difficulty with this approach is that the work factor calculations would probably differ depending upon which relation the analyst chose to regard as the critical factor. This complication would necessitate changing the operation data bank.

6. Results

The PROLOG system produces a report on every output relation from the given start onward, following first the main data flow path and then each branching path. The appearance of the report on a highly sensitive relation deserving maximal protection would be:

```
Total          = 200,
Workf          = 0,
X              = credit-sales + ratings,
Nonsensitive   = 0,
D              = 1,
O              = join;
```

The appearance of the report on a nonsensitive relation deserving little if any protection would be:

```
Total          = 33,
Workf          = 8,
X              = purchase-orders,
Nonsensitive   = 29,
D              = 7,
O              = split;
```

The appearance of the report on a moderately sensitive relation deserving moderate protection would be:

```
Total          = 297,
Workf          = 4,
X              = sales-invoice,
Nonsensitive   = 135,
D              = 9,
O              = calculate;
```

The appearance of a report on a more sensitive relation deserving a higher level of protection would be:

```
Total          = 53,
Workf          = 2,
X              = rejected-orders,
Nonsensitive   = 15,
D              = 7,
O              = union-split;
```

The analysis of our example data-processing system obtained from the PROLOG program is summarized in Table 2.

Table 2
Properties of Relations

No.	Name	Operation	I/O	I	N	WF
1	sales-order	validate(1)	I	360	NA	NA
2	error-A	validate(1)	O	10	NA	NA
3	valid-orders	validate(1)	O	350	NA	NA
4	template-A	validate(1)	I	50	NA	NA
3	valid-orders	validate(2)	I	350	NA	NA
5	credit-sales	validate(2)	O	200	0	0
6	cash-sales	validate(2)	O	150	150	10
7	credit/cash-criteria	validate(2)	I	10	10	10
5	credit-sales	join(1)	I	200	0	0
8	credit-file	join(1)	I	1000	0	0
9	credit-sales + ratings	join(1)	O	200	0	0
9	credit-sales + ratings	validate(3)	I	200	0	0
10	reject-sales-credit	validate(3)	O	20	0	0
11	good-credit-sales	validate(3)	O	180	0	0
12	credit-criteria	validate(3)	I	100	100	10
11	good-credit-sales	union + split(1)	I	180	0	0
13	credit/cash-flag	union + split(1)	O	330	150	4
14	good-sales	union + split(1)	O	330	150	4
6	cash-sales	union + split(1)	I	150	150	10
14	good-sales	join(2)	I	330	150	4
15	sales + quantity-on/hand	join(2)	O	330	150	4
16	inventory-file	join(2)	I	1000	1000	10
15	sales + quantity-on/hand	validate(4)	I	330	150	4
17	reject-sales-shortage	validate(4)	O	33	15	4
18	fillable-sales	validate(4)	O	297	135	4
19	shipping-criteria	validate(4)	I	100	100	10
17	reject-sales-shortage	split(1)	I	33	15	4
20	reject-sales-shortage(1)	split(1)	O	33	15	4
21	shortage	split(1)	I	33	29	9
21	shortage	split(2)	I	33	29	9
22	orders	split(2)	O	33	29	9
23	purchase-orders	split(2)	O	33	29	9
10	reject-sales-credit	union + split(2)	I	20	0	0
24	credit-cash-flag-A	union + split(2)	O	53	15	2
25	rejected-orders	union + split(2)	O	53	15	2
20	reject-sales-shortage(1)	union + split(2)	I	33	15	4
22	orders	update(1)	I	33	29	9
26	inventory-file(1)	update(1)	I	1000	1000	10
27	inventory-file + orders	update(1)	O	1033	1029	9
18	fillable-orders	split(3)	I	297	135	4
28	issues	split(3)	O	297	237	8
29	sales	split(3)	O	297	135	4
29	sales	calculate	I	297	135	4
30	sales + totals	calculate	O	297	135	4
30	sales + totals	split(4)	I	297	135	4
31	sales-report	split(4)	O	297	267	9
32	sales + totals(1)	split(4)	O	297	135	4
32	sales + totals(1)	calculate + select	I	297	135	4
33	sales-invoices	calculate + select	O	297	135	4
34	sales-tax-table	calculate + select	I	100	100	10
28	issues	update(2)	I	297	237	8
35	inventory-file + orders(1)	update(2)	I	1033	1029	9
36	inventory-file + orders + issues	update(2)	O	1330	1266	9

One can imagine an attack on this system in the "controlled plain text" model. An opponent who wants to learn the contents of "credit-file" could generate "sales-orders" exhaustively for all individuals about whose credit rating he was curious. He could then observe whether he received "rejected-orders" or "sales-invoice" messages. He would eventually learn whose credit was good and whose was bad, but with a margin of uncertainty. A subject may have received a "rejected-orders" message either because his credit was bad or because the firm was out of stock of the item ordered. Generating all these spurious orders would be costly and time consuming. That's where the concept of "work factor" comes in.

If the opponent is unable or unwilling to control the input, we would have the equivalent of a "known plain text" attack provided he could observe the input. He would, however, have to wait until his subject ordered merchandise. This approach would be less costly but could take a very long time indeed.

If the opponent is unable to observe or control input, he is in a position akin to a cryptanalyst. He faces the dual uncertainty as to whether a subject receiving a "rejected-orders" message had bad credit or ordered an out-of-stock item; and whether a subject receiving a "sales-invoice" had good credit or paid cash.

7. Conclusions

We believe the most significant part of this work is that it provides the systems analyst with a quantitative design tool. The PROLOG language is sufficiently flexible that other system parameters such as cost and time can be computed as well as work factor.

The algorithms for computing work factor are certainly open to challenge. They can, however, easily be modified to reflect different security requirements or assumptions regarding the environment.

We have applied this model successfully to several representative systems; a payroll system, in which the critical relation was the personnel-file interface containing each employee's pay code; a computer-aided-manufacturing system, in which the critical relation was the engineering file; and two operating-system programs controlling access by the ticket system in the first instance, and by an access-control list, in the second. In all cases, the analytical procedure was able to cope with configuration variations; and produced results that conformed to expectations.

Acknowledgements

The authors acknowledge the encouragement and assistance of T.E. Elcock in applying the PROLOG language; and the support of the National Scientific and Engineering Research Council (Canada) under grant #A-7132.

References

[1] R. Ennals, "An Introduction to PROLOG", Department of Computing, Imperial College, London.

[2] G.H. Mac Ewen, "The Design for a Secure System Based on Program Analysis", Technical Report No. 81-118, Department of Computer Science, Queen's University, Kingston, Ontario.

[3] G.H. Mac Ewen, "A Hierarchical Security Model for Program Certification Systems", Technical Report No. 81-116, Department of Computer Science, Queen's University, Kingston, Ontario.

[4] A.L. Mennie and G.H. Mac Ewen, Information Flow Certification Using an Intermediate Code Program Representation, *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 6, November 1981.

[5] F.C. Pereira and D.H.D. Warren, "Users' Guide to Dec-System-10 PROLOG", Department of Artificial Intelligence, University of Edinburgh.