Integrating the Structured Analysis and Design Models: an Initial Algebra Approach

T. H. Tse†

Numerous models have been proposed under the name of structured systems analysis and design. Because of the lack of a common theoretical framework, the transition from one model to another is arbitrary and can only be done manually. An initial algebra approach is proposed to integrate the structured models. The algebra defined can be mapped through homomorphisms to Yourdon structure charts and DeMarco data flow diagrams. It can also be linked to Jackson structure text through equations.

Keywords and Phrases: **algebraic approach, structured analysis, structured design**
CR Categories: **D.2.1, F.3.2, K.6.3**

1. INTRODUCTION

Numerous models have been proposed under the name of structured analysis and design. Examples are data flow diagrams (DeMarco, 1978; Gane and Sarson, 1979; Weinberg, 1980), Jackson structure diagrams, Jackson structure text (Jackson, 1975), system specification diagrams, system implementation diagrams (Jackson, 1983), Warnier/Orr diagrams (Orr, 1977) and structure charts (Yourdon and Constantine, 1979). They are widely accepted by practising systems analysts and designers through the simplicity of use and the ease of communication with users. But because of the lack of a common theoretical framework, the transition from one model to another is arbitrary and can only be done manually. Users tend to stick to a particular model not because of its superiority but because of familiarity. Automatic development aids tend to be *ad hoc* and model-dependent.

To solve the problem, there is a need to provide a formal link for the structured models. The initial algebra approach is proposed. It has a rich mathematical linkage with category and algebraic theories. But at the same time, the concepts can be simply stated for those who do not want to be involved with elaborate theories.

In this paper we will define the algebra and illustrate how it can be related to Yourdon structure charts, DeMarco data flow diagrams and Jackson structure text by means of homomorphisms and equations. The three models have been chosen for illustration because they represent three distinct classes of structured models.

We shall concentrate on the conceptual framework rather than on formal proofs. Only a knowledge of elementary set theory will be assumed. Readers who are interested in a deeper understanding of algebraic formalism may refer to Cohn (1981) for a general algebraic introduction, to Burstall and Goguen (1982), Goguen *et al.* (1978), Wagner (1981) and Zilles *et al.* (1982) for a computer science oriented treatment, and to Goguen *et al.* (1975) and Wagner *et al.* (1977) for a category-theoretic treatment.

2. ADVANTAGES OF INTEGRATION

An integration of the structured models is useful for several reasons:

a. Specifications can be transformed from one form to another through homomorphisms and equations, as illustrated in the subsequent sections of the paper.

b. Different structured models are suitable for different situations depending on the environment (Shigo *et al.*, 1980), emphasis (Colter, 1982) and stage of development (Lauber, 1982). But it has been found that individual models may not be used in some installations because the users are not familiar with them (Beck and Perkins, 1983). Through a transformation system, the most suitable model can be used independently of user

†*Centre of Computer Studies and Applications, University of Hong Kong, Pokfulam Road, Hong Kong.* *Manuscript received July, 1985; revised January, 1986.*

```
task (process-sales; get (); put ();
sequ (task (get-valid-order; get (); put (outdata valid-order);
      sequ (task (get-order; get (source customer); put (outdata order); elem);
            task (validate-order; get (indata order); put (outdata valid-order); elem)));
      task (process-order; get (indata valid-order); put (outdata invoice-info);
      seln (task (process-local-order; get (indata lc-order); put (outdata invoice-info);
            sequ (task (prepare-local-invoice;
                        get (indata lc-order); put (outdata pre-tax-info); elem);
                  task (compute-tax;
                        get (indata pre-tax-info); put (outdata invoice-info); elem)));
            task (prepare-overseas-invoice;
                  get (indata os-order); put (outdata invoice-info); elem)));
      task (put-invoice; get (indata invoice-info); put (outfile invoice); elem)))
```

Figure 1. Sample Term in Initial Algebra

familiarity.

c. Automatic development aids for one structured methodology can be applied to another through transformations. The development aids described, for example, in Delisle *et al.* (1982), DeMarco and Soceneantu (1984) and Tse (1985) may therefore be extended to other models.

d. In recent years the initial algebra approach has been used extensively in the specification of abstract data types. Examples are Clear (Burstall and Goguen, 1980, 1981; Sannella, 1984) and OBJ (Futatsugi *et al.*, 1985; Goguen 1984; Goguen and Tardo, 1979; Goguen *et al.*, 1983). Interpreters for abstract data types are already available. Although such interpreters are not originally intended for structured analysis and design models, they can nevertheless be adapted to suit our needs, e.g. for validating our specifications.

e. In informal specifications such as DeMarco data flow diagrams, a certain degree of omission or 'mutual understanding' is permitted. This often leads, however, to ambiguity and misunderstanding. If a formal specification is used, we can enforce predefined standards more easily.

3. ALGEBRAS

Intuitively, an algebra is a family of objects that satisfy a formal structure. To define an algebra A, we must first of all define the formal structure through the concept of a signature. A *signature* consists of a set S of object types, known as *sorts*, together with a family Σ of sets, each set containing *operation symbols* (or simply *symbols*) that connect the sorts. We will use $\Sigma<s_1 \ldots s_n, s>$ to denote the set of operation symbols that connect the sorts $s_1, \ldots, s_n$ to the sort s.

Given the skeleton structure, we then complete the definition by relating it to real objects. Each sort s is mapped to a set $A<s>$, which is called the *carrier* of s. Each symbol $\mathbf{q}$ in $\Sigma<s_1 \ldots s_n, s>$ is mapped to a function

$$\mathbf{q}_A: A<s_1> \times \ldots \times A<s_n> \rightarrow A<s>$$

which is called an *operation*.

Let us apply the algebraic fundamentals to structured systems. Conceptually, a structured system is specified by a hierarchy of tasks. Each task consists of a name, a structure, together with the input and output. The structure determines whether the task is elementary, or is made up of subtasks in the form of sequence, selection, iteration or parallelism. The input and output are in the form of data flows related with other tasks, files and the environment.

The signature for structured systems, then, consists of a set S of seven sorts: *task, name, struct, input, output, dataflow* and *flowname*, and a family Σ of sets of operation symbols:

$\Sigma<name, input, output, struct, task> = \{$ **task** $\}$
$\Sigma<task^n, struct> = \{$ **sequ, seln, para** $\}$
$\Sigma<task, struct> = \{$**iter** $\}$
$\Sigma<\Lambda, struct> = \{$ **elem** $\}$
$\Sigma<dataflow^n, input> = \{$ **get** $\}$
$\Sigma<dataflow^n, output> = \{$ **put** $\}$
$\Sigma<flowname, dataflow> = \{$ **indata, inflag, infile, source, outdata, outflag, outfile, sink** $\}$

for any positive integer n and where Λ is the empty string.

Table 1.

Sort	Carrier
task	set of tasks
name	set of task names
struct	set of structures
input	set of inputs
output	set of outputs
dataflow	set of data flows
flowname	set of data flow names

The sorts of the signature are mapped to the carriers as shown in Table 1. The symbols are mapped to the following operations:

a. The operation $\mathbf{task}_A$ specifies the name, structure, input and output of a task.

b. The operations $\mathbf{sequ}_A$, $\mathbf{seln}_A$, $\mathbf{iter}_A$ and $\mathbf{para}_A$ link up a number of subtasks to form a structure.

c. The operation $\mathbf{elem}_A$ indicates that a structure is elementary, i.e. it does not consist of subtasks.

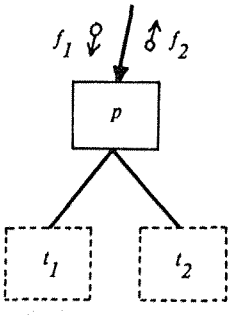
d. The operations $\mathbf{get}_A$ and $\mathbf{put}_A$ specify the input and output flows for a task.

e. The operations $\mathbf{indata}_A$, $\mathbf{inflag}_A$, $\mathbf{infile}_A$ and $\mathbf{source}_A$ specify the name of each input data flow. They also denote, respectively, that the data flow consists of data from some other task, a flag from some other task, data from a file, and data from the environment.

f. The operations $\mathbf{outdata}_A$, $\mathbf{outflag}_A$, $\mathbf{outfile}_A$ and $\mathbf{sink}_A$ are similarly used for output data flows.

Different algebras can be defined over the same signature. *Homomorphisms*, or functions preserving the signature, can be defined from one algebra to another. Such homomorphisms enable us to forget about minor syntactical differences in various specification methods and concentrate on the major issues. For example, suppose

$\mathbf{task}_A(p, \mathbf{get}_A(\mathbf{indata}_A(f_1)),$
$\mathbf{put}_A(\mathbf{outdata}_A(f_2)), \mathbf{sequ}_A(t_1, t_2)) =$



. . . (1)

in algebra A, and

$\mathbf{task}_B(p, \mathbf{get}_B(\mathbf{indata}_B(f_1)),$
$\mathbf{put}_B(\mathbf{outdata}_B(f_2)), \mathbf{sequ}_B(t_1, t_2)) =$

procedure $p(f_1, f_2)$: **begin** t_1; t_2 **end**. . . (2)

in algebra B. Then a homomorphism mapping the variables p, f_1, f_2, t_1 and t_2 in A to the corresponding variables in B will automatically map (1) to (2).

4. INITIAL ALGEBRA

The algebra that has the richest context is called an *initial algebra*, satisfying the following property:

> An algebra A is initial if, for any algebra B over the same signature, there exists a unique homomorphism mapping A to B.

We would like to construct an initial algebra for structured systems using the concept of term algebras. A *term algebra* T_Σ for structured systems is defined as follows:

4.1 Carriers

We regard task names and data names as more fundamental than other variables in our algebra, because these names appear unaltered in a final specification. They are like terminals in the theory of formal languages. We will enlarge the signature by putting in task names and data names as 'symbols'. Thus two more sets of symbols are defined:

$\Sigma<\Lambda, \mathit{name}>$ = the set of task names,
$\Sigma<\Lambda, \mathit{flowname}>$ = the set of names of input/output data flows.

Although also known as 'operation symbols', these symbols are actually not operating on any sort.

Let X denote the enlarged set of symbols, together with three delimiter symbols: '(', ';' and ')'. The carriers of T_Σ are made up of *terms* in X, i.e. strings of symbols from X. We define the carriers $T_\Sigma<s>$ by induction as follows:

a. For any symbol $\mathbf{q}$ in $\Sigma<\Lambda, s>$, we let the term '$\mathbf{q}$' be in $T_\Sigma<s>$.

b. For any operation symbol $\mathbf{q}$ in $\Sigma<s_1 \ldots s_n, s>$, and for any terms '$u_1$' in $T_\Sigma<s_1>$, . . . , 'u_n' in $T_\Sigma<s_n>$, we let the term '$\mathbf{q}(u_1$; . . . ; $u_n)$' be in $T_\Sigma<s>$.

4.2 Operations

Operations $\mathbf{q}_T$ in T_Σ are induced from the symbols $\mathbf{q}$ as follows:

a. For any symbol $\mathbf{q}$ in $\Sigma<\Lambda, s>$, we define $\mathbf{q}_T$ to be the term '$\mathbf{q}$'.

b. For any operation symbol $\mathbf{q}$ in $\Sigma<s_1 \ldots s_n, s>$, and for any terms '$u_1$' in $T_\Sigma<s_1>$, . . . , 'u_n' in $T_\Sigma<s_n>$, we define $\mathbf{q}_T(u_1, \ldots, u_n)$ to be the term '$\mathbf{q}(u_1; \ldots; u_n)$'.

It can be shown that the term algebra T_Σ thus defined is an initial algebra. It can be mapped by homomorphisms to other algebras over the same signature. For example, we will illustrate how the sample term shown in Figure 1 can be related to a Yourdon structure chart, a DeMarco data flow diagram and Jackson structure text.

5. YOURDON STRUCTURE CHARTS

To illustrate how the terms in our initial algebra can be mapped to structure charts, we must first of all define an algebra Y of Yourdon structure charts (which we will call *Yourdon algebra* for short). The carriers are defined similarly to Table 1. The operations $\mathbf{q}_Y$ are defined as shown in Figure 2. Then the obvious homomorphism will map our initial algebra to the Yourdon algebra. The term in Figure 1, for example, will be mapped to the structure chart of Figure 3.

Can we do the reverse? That is to say, can we define a unique reverse homomorphism from the Yourdon algebra to the initial algebra, and hence get back our term? We find that even though reverse homomorphisms can be created, they are not unique. One reason is that some operations, such as $\mathbf{infile}_Y$, $\mathbf{outfile}_Y$, $\mathbf{source}_Y$ and $\mathbf{sink}_Y$, are effectively not used

$\mathbf{task}_Y$: $Y\langle name\rangle \times Y\langle input\rangle \times Y\langle output\rangle \times Y\langle struct\rangle \rightarrow Y\langle task\rangle$

$\mathbf{sequ}_Y$: $Y\langle task\rangle^n \rightarrow Y\langle struct\rangle$

$\mathbf{seln}_Y$: $Y\langle task\rangle^n \rightarrow Y\langle struct\rangle$

$\mathbf{para}_Y$: $Y\langle task\rangle^n \rightarrow Y\langle struct\rangle$

$\mathbf{iter}_Y$: $Y\langle task\rangle \rightarrow Y\langle struct\rangle$

$\mathbf{elem}_Y$: $\rightarrow Y\langle struct\rangle$

$\mathbf{get}_Y$: $Y\langle dataflow\rangle^n \rightarrow Y\langle input\rangle$

$\mathbf{put}_Y$: $Y\langle dataflow\rangle^n \rightarrow Y\langle output\rangle$

$\mathbf{indata}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

$\mathbf{inflag}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

$\mathbf{infile}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

$\mathbf{source}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

$\mathbf{outdata}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

$\mathbf{outflag}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

$\mathbf{outdata}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

$\mathbf{sink}_Y$: $Y\langle flowname\rangle \rightarrow Y\langle dataflow\rangle$

such that

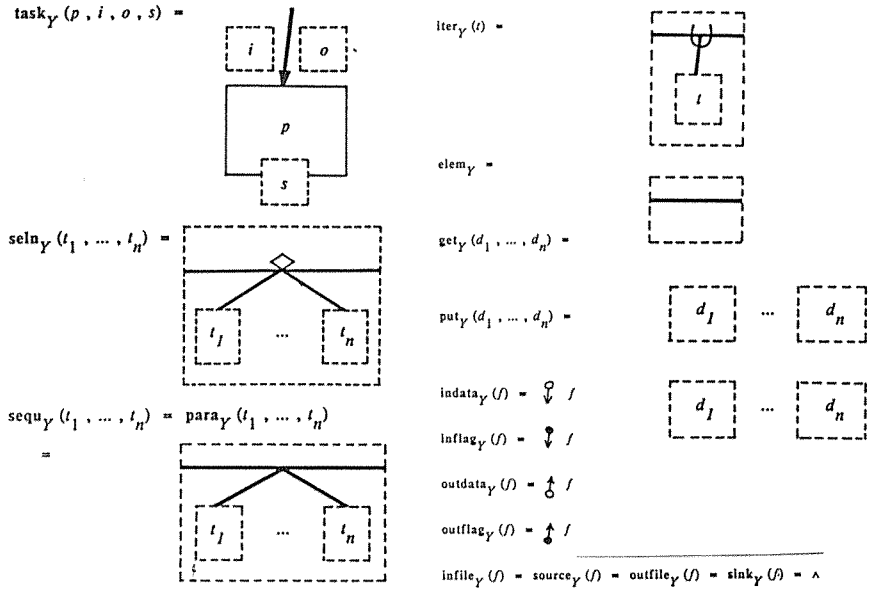


Figure 2. Operations in Yourdon Algebra

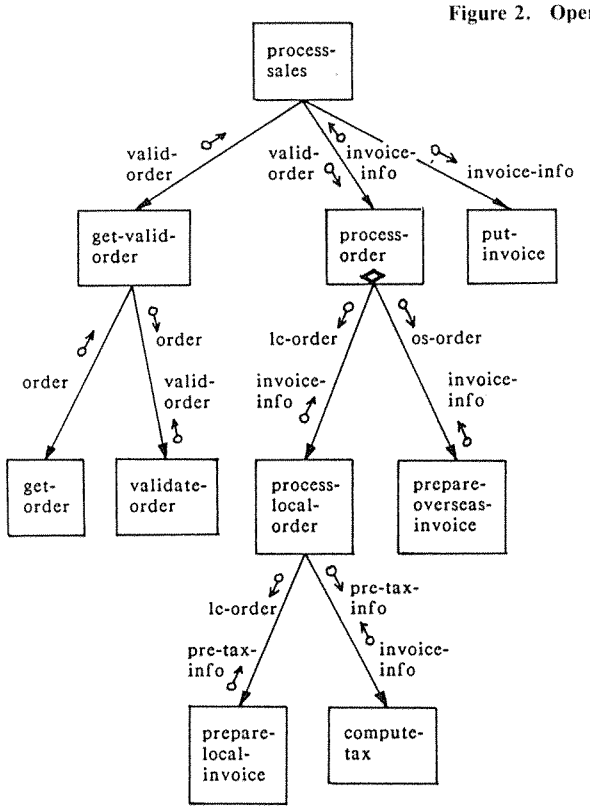


Figure 3. Sample Structure Charts in Yourdon Algebra

in the Yourdon algebra. Another reason is that other operations, although distinct in the initial algebra, may overlap in the Yourdon algebra. For example, both $\mathbf{sequ}_Y$ and $\mathbf{para}_Y$ give identical results. The Yourdon algebra must be extended to cater for these operations before a unique reverse homomorphism can be defined. The algebra of *extended* Yourdon structure charts will then be regarded as isomorphic, or equivalent, to our term algebra.

A transformation system is being studied by the present author to provide a computerized link between the initial algebra and the Yourdon algebra. It will be run using Smalltalk under Unix. It will accept any term in our initial algebra as input and generate a Yourdon structure chart automatically. Conversely, the system will also accept a Yourdon structure chart, enquire of the user about any extension required, and then convert the chart into a term in the initial algebra.

6. DeMARCO DATA FLOW DIAGRAMS

We can similarly define an algebra D of DeMarco data flow diagrams (or *DeMarco algebra* for short). The carriers are defined similarly to Table 1, and the operations are as shown in Figure 4. The obvious homomorphism will map the terms in our initial algebra to data flow diagrams. The term in Figure 1, for instance, can be mapped to the DeMarco data flow diagram of Figure 5. Furthermore, if we forget about the intermediate task names in the original term, then a flattened data flow diagram can be obtained, as shown in Figure 6.

The transformation system under study (see Section 5) will also accept terms in the initial algebra and generate DeMarco data flow diagrams automatically. Conversely, given a data flow diagram, the system will enquire the user about extensions to the DeMarco algebra and then generate automatically a term of the initial algebra. The system will therefore help the user to convert DeMarco data flow diagrams into Yourdon structure charts.

7. JACKSON STRUCTURE TEXT

We also want to define an algebra J of Jackson structure text (or *Jackson algebra*). This is done through the algebraic concept of equations, and involves the

$\mathbf{task}_D$: $D\langle name\rangle \times D\langle input\rangle \times D\langle output\rangle \times D\langle struct\rangle \rightarrow D\langle task\rangle$
$\mathbf{sequ}_D$: $D\langle task\rangle^n \rightarrow D\langle struct\rangle$
$\mathbf{seln}_D$: $D\langle task\rangle^n \rightarrow D\langle struct\rangle$
$\mathbf{para}_D$: $D\langle task\rangle^n \rightarrow D\langle struct\rangle$
$\mathbf{iter}_D$: $D\langle task\rangle \rightarrow D\langle struct\rangle$
$\mathbf{elem}_D$: $\rightarrow D\langle struct\rangle$
$\mathbf{get}_D$: $D\langle dataflow\rangle^n \rightarrow D\langle input\rangle$
$\mathbf{put}_D$: $D\langle dataflow\rangle^n \rightarrow D\langle output\rangle$
$\mathbf{indata}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$
$\mathbf{inflag}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$
$\mathbf{infile}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$
$\mathbf{source}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$
$\mathbf{outdata}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$
$\mathbf{outflag}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$
$\mathbf{outdata}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$
$\mathbf{sink}_D$: $D\langle flowname\rangle \rightarrow D\langle dataflow\rangle$

such that

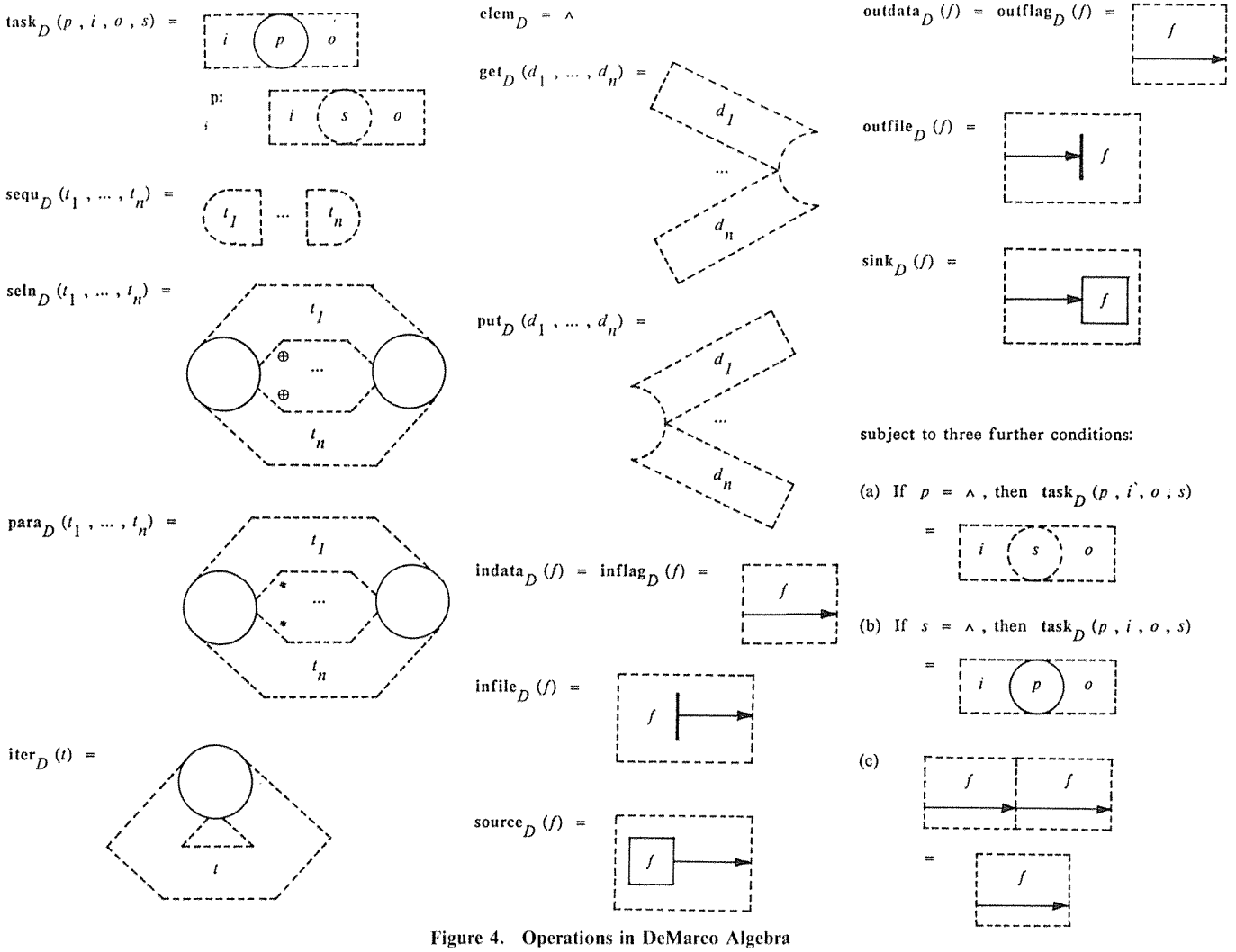


Figure 4. Operations in DeMarco Algebra

following steps:

a. We define a preliminary Jackson algebra which has the same carrier and operations as our initial algebra.

b. We define four new operations

$\mathbf{seq}_J$: $J\langle name\rangle \times J\langle task\rangle^n \rightarrow J\langle struct\rangle$
$\mathbf{sel}_J$: $J\langle name\rangle \times J\langle task\rangle^n \rightarrow J\langle struct\rangle$
$\mathbf{itr}_J$: $J\langle name\rangle \times J\langle task\rangle \rightarrow J\langle struct\rangle$
$\mathbf{elm}_J$: $J\langle name\rangle \rightarrow J\langle struct\rangle$.

The functions of these operations are shown in Figure 7.

c. Since input and output are not included in Jackson structure text, we must define two *equations*, or relationships that connect different elements of the algebra together. Thus we have

$\mathbf{get}_J(\wedge) = \mathbf{get}_J(d_1, \ldots, d_n)$
$\mathbf{put}_J(\wedge) = \mathbf{put}_J(d_1, \ldots, d_n)$

for any data flows $d_1, \ldots, d_n$.

d. We define four more equations to link up the new operations in (b) with the original operations, thus:

$\mathbf{seq}_J(p, t_1, \ldots, t_n) =$
$\mathbf{task}_J(p, \mathbf{get}_J(\wedge), \mathbf{put}_J(\wedge), \mathbf{sequ}_J(t_1, \ldots, t_n))$

$\mathbf{sel}_J(p, t_1, \ldots, t_n) =$
$\quad \mathbf{task}_J(p, \mathbf{get}_J(\wedge), \mathbf{put}_J(\wedge), \mathbf{seln}_J(t_1, \ldots, t_n))$
$\mathbf{itr}_J(p, t) = \mathbf{task}_J(p, \mathbf{get}_J(\wedge), \mathbf{put}_J(\wedge), \mathbf{iter}_J(t))$
$\mathbf{elm}_J(p) = \mathbf{task}_J(p, \mathbf{get}_J(\wedge), \mathbf{put}_J(\wedge), \mathbf{elem}_J)$

for any task names p and tasks $t, t_1, \ldots, t_n$.

In this way, the obvious homomorphism will enable us to map our terms into Jackson text. For example, the term in Figure 1 will be mapped to the Jackson structure text of Figure 8. Furthermore, the transformation system under study can also be enhanced to accept or generate Jackson structure text.

8. CONCLUSION

Structured analysis and design models can be integrated algebraically. A term algebra has been defined and can be mapped through homomorphisms to Yourdon structure charts and DeMarco data flow diagrams. It can also be linked to Jackson structure text through equations.

As a result, specifications can be transformed from one form to another. The most suitable model can be chosen for a target system independently of user familiarity. Algebraic interpreters may be adapted to validate the specifications. Automatic development aids for one methodology may be applied to another.

ACKNOWLEDGEMENTS

Part of this research was done at the London School of Economics, University of London under a Commonwealth Academic Staff Scholarship. The author is indebted to Ronald Stamper, Haya Freedman and Professor Frank Land of LSE for some invaluable suggestions. He is also grateful to Professors Joseph Goguen of Stanford, Jim Emery of Pennsylvania, Blake Ives of Dartmouth and Joe Davis of Indianna for the most encouraging comments on the project.

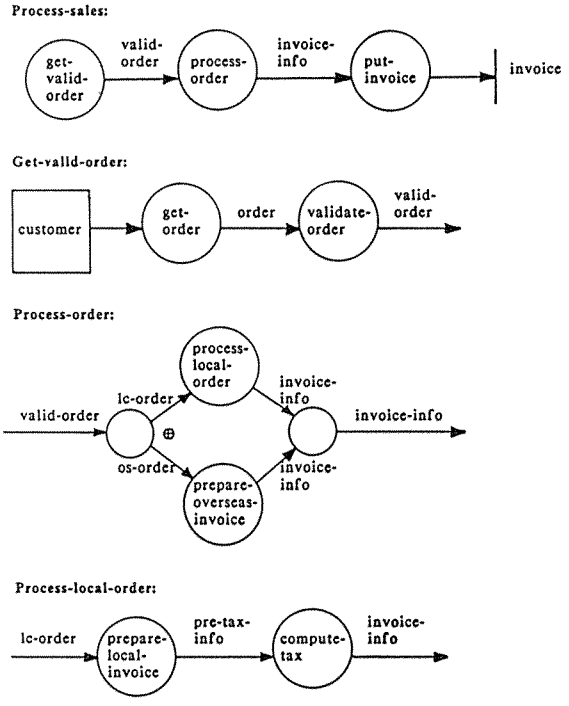


Figure 5. Sample Data Flow Diagram in DeMarco Algebra

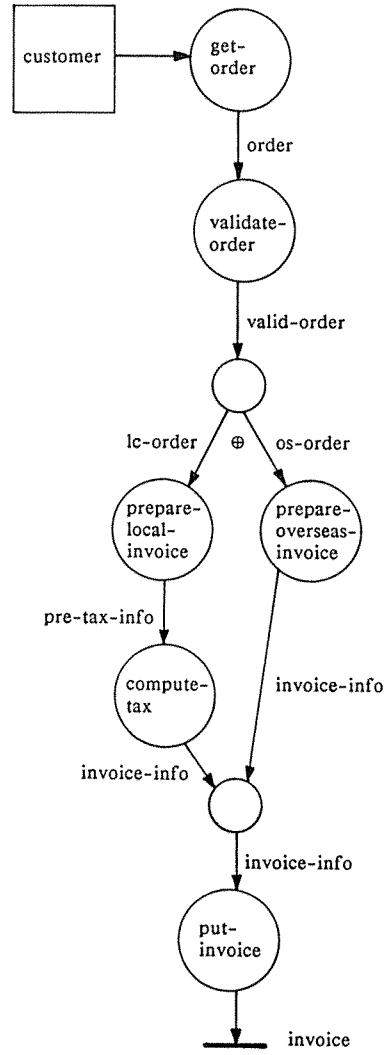


Figure 6. Further Data Flow Diagram in DeMarco Algebra

$\mathbf{seq}_J(p, t_1, \ldots, t_n) =$ p **seq** t_1 ; ... ; t_n ; p **end**

$\mathbf{sel}_J(p, t_1, \ldots, t_n) =$ p **sel** t_1 ; p **alt** ... ; p **alt** t_n ; p **end**

$\mathbf{itr}_J(p, t) =$ p **itr** t p **end**

$\mathbf{elm}_J =$ p

Figure 7. New Operations in Jackson Algebra

```
process-sales seq
    get-valid-order seq
        get-order;
        validate-order
    get-valid-order end;
    process-order sel
        process-local-order seq
            prepare-local-invoice;
            compute-tax
        process-local-order end;
    process-order alt
        prepare-overseas-invoice
    process-order end;
    put-invoice
process-sales end
```

Figure 8. Sample Structure Text in Jackson Algebra

References

BECK, L.L. and PERKINS, T.E. (1983): A survey of software engineering practice: tools, methods and results, *IEEE Transactions on Software Engineering, vol. SE-9, no. 5*, pp. 541-561.

BURSTALL, R.M. and GOGUEN, J.A. (1980): *The semantics of Clear, a specification language*, Lecture Notes in Computer Science, vol. 86, Springer-Verlag, Berlin, pp. 292-332.

BURSTALL, R.M. and GOGUEN, J.A. (1981): 'An informal introduction to specifications using Clear', in *Correctness Problem in Computer Science*, Boyer, R. and Moore, J. (eds.), Academic Press, London, pp. 185-213.

BURSTALL, R.M. and GOGUEN, J.A. (1982): 'Algebras, theories and freeness: an introduction for computer scientists', in *Theoretical Foundations of Programming Methodology*, Broy, M. and Schmidt, G. (eds.), C. Reidel, Dordrecht, Holland.

COHN, P.M. (1981): *Universal Algebra*, C. Reidel, Dordrecht, Holland.

COLTER, M.A. (1982): 'Evolution of the structured methodologies', in *Advanced System Development/Feasibility Techniques*, Couger, J.D., Colter, M.A. and Knapp, R.W. (eds.), Wiley, New York.

DELISLE, N.M., MENICOSY, D.E. and KERTH, N.L. (1982): 'Tools for supporting structured analysis', in *Automated Tools for Information Systems Design*, Schneider, H.-J. and Wasserman, A.I. (eds.), North-Holland, Amsterdam, pp. 11-20.

DEMARCO, T. (1978): *Structured Analysis and Systems Specification*, Prentice-Hall, Englewood Cliffs, New Jersey.

DEMARCO, T. and SOCENEANTU, A. (1984): 'SYNCRO: a data flow command shell for the Lilith/Modula computer', in *Proceedings of 7th International Conference on Software Engineering*, IEEE, New York, pp. 207-213.

FUTATSUGI, K., GOGUEN, J.A., JOUANNAUD, J.-P. and MESEGUER, J. (1985): 'Principles of OBJ2', in *Proceedings of Principles of Programming Symposium*, Association for Computing Machinery, New York.

GANE, C. and SARSON, T. (1979): *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Englewood Cliffs, New Jersey.

GOGUEN, J.A. (1984): Parameterized programming, *IEEE Transactions on Software Engineering, vol. SE-10, no. 5*, pp. 528-543.

GOGUEN, J.A. and TARDO, J.J. (1979): 'An introduction to OBJ: a language for writing and testing formal algebraic program specifications', in *Proceedings of Conference on Specification of Reliable Software*, IEEE, New York, pp. 170-189.

GOGUEN, J.A., MESEGUER, J. and PLAISTED, D. (1983): 'Programming with parameterized abstract objects in OBJ', in *Theory and Practice of Software Technology*, Ferrari, D. and Goguen, J.A. (eds.), North-Holland, Amsterdam, pp. 163-193.

GOGUEN, J.A., THATCHER, J.W., WAGNER, E.G. and WRIGHT, J.B. (1975): *An introduction to categories, algebraic theories and algebras*, Research Report RC 5369, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

GOGUEN, J.A., THATCHER, J.W. and WAGNER, E.G. (1978): 'An initial algebra approach to specification, correctness and implementation of abstract data types', in *Data Structuring, Current Trends in Programming Methodology*, vol. IV, Yeh, R.T. (ed.), Prentice-Hall, Englewood Cliffs, New Jersey, pp. 80-149.

JACKSON, M.A. (1975): *Principles of Program Design*, Academic Press, London.

JACKSON, M.A. (1983): *System Development*, Prentice-Hall, Englewood Cliffs, New Jersey.

LAUBER, R.J. (1982): Development support systems, *IEEE Computer, vol. 15, no. 5*, pp. 36-46.

ORR, K.T. (1977): *Structured Systems Development*, Yourdon, New York.

SANNELLA, D. (1984): A set-theoretic semantics for Clear, *Acta Informatica, vol. 21*, pp. 443-472.

SHIGO, O., IWAMOTO, K. and FUJIBAYASHI, S. (1980): A software design system based on a unified design methodology, *Journal of Information Processing, vol. 3, no. 3*, pp. 186-196.

TSE, T.H. (1985): An automation of Jackson's structured programming, *Australian Computer Journal, vol. 17, no. 4*, pp. 154-162.

WAGNER, E.G. (1981): *Lecture notes on the algebraic specification of data types*, Research Report RC 9203, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

WAGNER, E.G., THATCHER, J.W. and WRIGHT, J.B. (1977): *Free continuous theories*, Research Report RC 6906, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

WEINBERG, V. (1980): *Structured Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

YOURDON, E. and CONSTANTINE, L.L. (1979): *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*, Prentice-Hall, Englewood Cliffs, New Jersey.

ZILLES, S.N., LUCAS, P. and THATCHER, J.W. (1982): *A look at algebraic specifications*, Research Report RJ 3568, IBM San Jose Research Laboratory, San Jose, California.

Biographical Note

T.H. Tse received his B.Sc. degree from the University of Hong Kong in 1970, and his M.Sc. degree from the University of London in 1979. He is currently a lecturer in computer science at the University of Hong Kong. His research interests include software engineering and formal methods in information systems.

Mr Tse is a member of the British Computer Society, the British Institute of Management and the Institute of Data Processing Management. He is a council member of the Vocational Training Council in Hong Kong. He was awarded an MBE by the Queen in 1982.