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

T.H. Tse
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
Email: thtse@cs.hku.hk

ABSTRACT

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, 1979; 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.

* © 1986 Australian Computer Journal. This material is presented to ensure timely dissemination of scholarly and technical work. Personal use of this material is permitted. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each authors copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. Permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from Australian Computer Journal.

† Part of this research was done at the London School of Economics, University of London under a Commonwealth Academic Staff Scholarship.
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

A 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 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 \( \Sigma \) of sets, each set containing operation symbols (or simply symbols) which connect the sorts. We will use \( \Sigma<s_1, \ldots, s_n, s> \) to denote the set of operation symbols which 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 \( q \) in \( \Sigma<s_1, \ldots, s_n, s> \) is mapped to a function

2
\[ q_A: A^{s_1} \times \ldots \times A^{s_n} \rightarrow A^{s}, \]

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:

\[ \text{task}, \text{name}, \text{struct}, \text{input}, \text{output}, \text{dataflow}, \text{flowname} \]

and a family \( \Sigma \) of sets of operation symbols:

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

for any positive integer \( n \) and where \( \Lambda \) is the empty string.

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 \( \text{task}_A \) specifies the name, structure, input and output of a task.
(b) The operations \( \text{sequ}_A, \text{seln}_A, \text{iter}_A \) and \( \text{para}_A \) link up a number of subtasks to form a structure.
(c) The operation \( \text{elem}_A \) indicates that a structure is elementary, i.e. it does not consist of subtasks.
(d) The operations \( \text{get}_A \) and \( \text{put}_A \) specify the input and output flows for a task.
(e) The operations \( \text{indata}_A, \text{inflag}_A, \text{infile}_A \) and \( \text{source}_A \) 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 \( \text{outdata}_A, \text{outflag}_A, \text{outfile}_A \) and \( \text{sink}_A \) 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
task$_A(p, \text{get}_A(\text{indata}_A(f_1)), \text{put}_A(\text{outdata}_A(f_2)), \text{sequ}_A(t_1, t_2)) = \ldots (1)

in algebra $A$, and

\[
\text{task}_B(p, \text{get}_B(\text{indata}_B(f_1)), \text{put}_B(\text{outdata}_B(f_2)), \text{sequ}_B(t_1, t_2)) = \text{procedure } p(f_1, f_2): \begin{align*}
t_1; t_2 \end{align*} \end{procedure} \ldots (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 $T_\Sigma$ 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:

$\Sigma_<\Lambda, \text{name}> = \text{the set of task names},$

$\Sigma_<\Lambda, \text{flowname}> = \text{the set of names of input/output data flows}.$

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 $T_\Sigma$ 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 $q$ in $\Sigma_<\Lambda, s>$, we let the term “$q$” be in $T_\Sigma<s>$.

(b) For any operation symbol $q$ 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 “$q(u_1; \ldots; u_n)$” be in $T_\Sigma<s>$.

4.2 Operations

Operations $q_T$ in $T_\Sigma$ are induced from the symbols $q$ as follows:

(a) For any symbol $q$ in $\Sigma_<\Lambda, s>$, we define $q_T$ to be the term “$q$"."
(b) For any operation symbol \( q \) in \( \Sigma^<s_1 \ldots s_n, s> \), and for any terms \( \langle u_1 \rangle \) in \( T_\Sigma^<s_1, \ldots, \langle u_n \rangle \), we define \( q(u_1; \ldots; u_n) \) to be the term \( \langle q(u_1; \ldots; u_n) \rangle \).

It can be shown that the term algebra \( T_\Sigma \) 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 \( q_Y \) 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 \( \text{infile}_Y, \text{outfile}_Y, \text{source}_Y \) and \( \text{sink}_Y \), are effectively not used 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 \( \text{sequ}_Y \) and \( \text{para}_Y \) 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 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 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

\[
\begin{align*}
\text{seq}_J &: J<\text{name}> \times J<\text{task}>^n \rightarrow J<\text{struct}> \\
\text{sel}_J &: J<\text{name}> \times J<\text{task}>^n \rightarrow J<\text{struct}> \\
\text{itr}_J &: J<\text{name}> \times J<\text{task}> \rightarrow J<\text{struct}> \\
\text{elm}_J &: J<\text{name}> \rightarrow J<\text{struct}>
\end{align*}
\]

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

\[
\begin{align*}
\text{get}_J(\Lambda) &= \text{get}_J(d_1, \ldots, d_n) \\
\text{put}_J(\Lambda) &= \text{put}_J(d_1, \ldots, d_n)
\end{align*}
\]

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:

\[
\begin{align*}
\text{seq}_J(p, t_1, \ldots, t_n) &= \text{task}_J(p, \text{get}_J(\Lambda), \text{put}_J(\Lambda), \text{sequ}_J(t_1, \ldots, t_n)) \\
\text{sel}_J(p, t_1, \ldots, t_p) &= \text{task}_J(p, \text{get}_J(\Lambda), \text{put}_J(\Lambda), \text{seln}_J(t_1, \ldots, t_n)) \\
\text{itr}_J(p, t) &= \text{task}_J(p, \text{get}_J(\Lambda), \text{put}_J(\Lambda), \text{iter}_J(t)) \\
\text{elm}_J(p) &= \text{task}_J(p, \text{get}_J(\Lambda), \text{put}_J(\Lambda), \text{elem}_J)
\end{align*}
\]

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

The author is indebted to Ronald Stamper, Haya Freedman and Professor Frank Land of LSE for their invaluable suggestions. He is also grateful to Professors Jim Emery of Philadelphia, Blake Ives of Dartmouth and Joe Davis of Indiana for the most encouraging comments on the project.

REFERENCES


Table 1.

<table>
<thead>
<tr>
<th>Sort</th>
<th>Carrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>set of tasks</td>
</tr>
<tr>
<td>name</td>
<td>set of task names</td>
</tr>
<tr>
<td>struct</td>
<td>set of structures</td>
</tr>
<tr>
<td>input</td>
<td>set of inputs</td>
</tr>
<tr>
<td>output</td>
<td>set of outputs</td>
</tr>
<tr>
<td>dataflow</td>
<td>set of data flows</td>
</tr>
<tr>
<td>flowname</td>
<td>set of data flow names</td>
</tr>
</tbody>
</table>
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);
    sele (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 co-order); put (outdata invoice-info); elem));
  task (put-invoice; get (indata invoice-info); put (outfile invoice); elem)));

Figure 1. Sample Term in Initial Algebra
such that

Figure 2. Operations in Yourdon Algebra
Figure 3. Sample Structure Charts in Yourdon Algebra
such that

\[
\text{task}_D(p, i, o, s) = \begin{array}{c}
\text{elem}_D = \lambda \\
\text{get}_D(d_1, \ldots, d_n) = \\
\text{put}_D(d_1, \ldots, d_n) = \\
\text{source}_D(f) = \\
\text{sink}_D(f) = \\
\text{iter}_D(f) = \\
\end{array}
\]

subject to three further conditions:

(a) If \( p = \lambda \), then \( \text{task}_D(p, i', o, s) = \lambda \)

(b) If \( s = \lambda \), then \( \text{task}_D(p, i, o, s) = \lambda \)

(c) \[
\begin{array}{c}
\text{sequ}_D(t_1, \ldots, t_n) = \\
\text{par}_D(t_1, \ldots, t_n) = \\
\text{indata}_D(f) = \text{inflag}_D(f) = \\
\text{inflle}_D(f) = \\
\end{array}
\]

\[\text{outdata}_D(f) = \text{outflag}_D(f) = \]

Figure 4. Operations in DeMarco Algebra
Figure 5. Sample Data Flow Diagram in DeMarco Algebra
Figure 6. Further Data Flow Diagram in DeMarco Algebra
Figure 7. New Operations in Jackson Algebra

\[\text{seq}_f(p, t_1, \ldots, t_n) = \begin{array}{l}
p \text{ seq} \\
\quad t_1 \\
\quad \vdots \\
\quad t_n \\
p \text{ end}
\end{array}\]

\[\text{sel}_f(p, t_1, \ldots, t_n) = \begin{array}{l}
p \text{ sel} \\
\quad t_1 \\
p \quad \text{alt} \\
\quad \vdots \\
p \quad \text{alt} \\
\quad t_n \\
p \text{ end}
\end{array}\]

\[\text{itr}_f(p, t) = \begin{array}{l}
p \text{ itr} \\
p \text{ end}
\end{array}\]

\[\text{elm}_f = \begin{array}{l}
p \\
\end{array}\]
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