A UNIFIED ALGEBRAIC VIEW OF STRUCTURED SYSTEMS DEVELOPMENT MODELS

T.H. Tse *
London School of Economics
University of London

1. INTRODUCTION

Numerous models have been proposed under the name of structured systems development. Examples are data flow diagrams [7, 10], Jackson structure diagrams, Jackson structure text [14], system specification diagrams, system implementation diagrams [15], Warnier/Orr diagrams [17], and Yourdon structure charts [22]. They are widely accepted by practising systems developers through the simplicity of use and the ease of communication with users. But because of the lack of a common theoretical framework, 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/theoretical link for the structured models. In the proposed dissertation, an initial algebra approach [3, 4, 13] is used. An algebra will be defined and linked to DeMarco data flow diagrams, Jackson structure text, and Yourdon structure charts. These three have been chosen because they represent three distinct forms of structured systems development models.

2. ADVANTAGES OF A UNIFIED ALGEBRAIC VIEW

A unified algebraic view of the structured models is useful for several reasons:

(a) Specifications can be transformed from one form to another through signature morphisms, equations, and derived operation symbols.

(b) Different structured models are suitable for different situations depending on the environment [19], emphasis [5], and stage of development [16]. But it has been found that individual models may not be used in some installations because the users are not familiar with them [1]. With a transformation system, the most suitable model can be used, independently of user familiarity.

(c) In recent years the initial algebra approach has been used extensively in the specification of abstract data types. Examples are Clear [2, 18] and OBJ [9, 11]. Although they are not originally intended for structured models, the interpreters for abstract data types can be adapted for validation of our specifications and the transformation between different models.

(d) Automatic development aids for one structured methodology can be applied to another method through transformations. The development aids described in [6], [8], and [20], for example, may be extended to other structured models.

---

* Current address: Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong.

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* that connect the sorts. We shall 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 **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 and a structure, together with the interfaces. The structure indicates whether the task is elementary, or is made up of subtasks in the form of sequence, selection, iteration or parallelism. The interfaces are in the form of data flows, which are input/output data related with other tasks, files, and the environment.

The signature for structured systems, then, consists of a set *S* of six sorts: *task, name, struct, inter, flow, data,* and a family Σ of sets of operation symbols:

$\Sigma<name\ inter\ struct^n, task>$ = {**task**}
$\Sigma<task^n, struct>$ = {**seq**, **sel**, **par**}
$\Sigma<task, struct>$ = {**itr**}
$\Sigma<\Lambda, struct>$ = {**elem**}
$\Sigma<flow^n, inter>$ = {**io**}
$\Sigma<data, flow>$ =
{**in**, **out**, **inflag**, **outflag**, **infile**, **outfile**, **source**, **sink**}

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

The sorts of the signature are mapped, respectively, to the set of tasks, the set of task names, the set of structures, the set of interfaces, the set of data flows, and the set of data names. The symbols are mapped to operations explained as follows:

(a) The operation $\mathbf{task}_A$ specifies the name, structure and interfaces of a task.

(b) The operations $\mathbf{seq}_A$, $\mathbf{sel}_A$, $\mathbf{itr}_A$, and $\mathbf{par}_A$ link up a number of subtasks into a structure.

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

(d) The operation $\mathbf{io}_A$ indicates that the interfaces are made up of a number of data flows.

(e) The input operations $\mathbf{in}_A$, $\mathbf{inflag}_A$, $\mathbf{infile}_A$, and $\mathbf{source}_A$ are used to name the data passing through a data flow into a task. They also indicate, respectively, that the data are inputs from some other task, flags from some other task, inputs from a file, and inputs from the

environment. The output operations $\mathbf{out}_A$, $\mathbf{outflag}_A$, $\mathbf{outfile}_A$, and $\mathbf{sink}_A$ are used similarly.

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 and concentrate on the major issues in a specification. For example, an expression like "**seq**(a; b; c)" in a particular algebra may be mapped to an alternative algebra giving

a **seq**
    b; c
a **end**

which is in the notation of Jackson structure text. Furthermore, if we allow operation symbols to vary through signature morphisms and/or derivation of symbols, the expressions can be mapped to more graphic forms giving DeMarco data flow diagrams or Yourdon structure charts.

4. INITIAL ALGEBRA

The algebra that has the richest context is called an *initial algebra*, denoted by A_0. It has the property that, given any other algebra A over the same signature, there exists a unique homomorphism mapping A_0 to A. Because of this guarantee of homomorphisms, we would like to use the initial algebra approach to link up various structured systems development models. A initial algebra for structured models is defined thus:

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 the final specification. We shall *enlarge* the signature by putting in task names and data names as *elementary symbols*.

Let X denote the enlarged set consisting of all operation symbols, elementary symbols, as well as three delimiter symbols: "(", ";", and ")". The carriers of A_0 are made up of *terms* in X, i.e. strings of symbols from X. We define the carriers $A_0\langle s\rangle$ by induction as follows:

(a) For any elementary symbol **q**, if it is a task name, we let the term "**q**" be in $A_0\langle name\rangle$, otherwise we let the term "**q**" be in $A_0\langle data\rangle$.

(b) For any operation symbol **q** in $\Sigma\langle s_1 \ldots s_n, s\rangle$, and for any terms u_1 in $A_0\langle s_1\rangle$, ..., u_n in $A_0\langle s_n\rangle$, we let the term "$\mathbf{q}(u_1; \ldots; u_n)$" be in $A_0\langle s\rangle$.

4.2 Operations

Operations $\mathbf{q}_{A_0}$ in A_0 are induced from the symbols **q** as follows:

(a) For any elementary symbol **q**, we define $\mathbf{q}_{A_0}$ to be the term "**q**".

(b) For any operation symbol **q** in $\Sigma\langle s_1 \ldots s_n, s\rangle$, and for any terms "$u_1$" in $A_0\langle s_1\rangle$, ..., "u_n" in $A_0\langle s_n\rangle$, we define $\mathbf{q}_{A_0}(u_1, \ldots, u_n)$ to be the term "$\mathbf{q}(u_1; \ldots; u_n)$".

It can be shown that the algebra A_0 thus defined is an initial algebra. It can be mapped by homomorphisms to other algebras over the same signature. Furthermore, if we allow signatures to vary, more user-friendly forms will result.

5. PROPOSED RESEARCH

It is proposed that structured models such as Yourdon structure charts, DeMarco data flow diagrams, and Jackson structure text be linked up using homomorphisms, signature morphisms, equations, and derived operation symbols. Specifications can thus be transformed from one form to another. A prototype system is further proposed to test the feasibilities. As a result of the research, the most suitable model may be chosen for a target system, independently of user familiarity. Algebraic interpreters may be used to validate the specifications. Automatic development aids for one methodology may be applied to another.

REFERENCES

[1] 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.

[2] 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.

[3] Burstall, R.M. and Goguen, J.A. (1982): Algebras, theories and freeness: an introduction for computer scientists, *Theoretical Foundations of Programming Methodology*, Broy, M. and Schmidt, G. (eds.), C. Reidel, Dordrecht, Holland.

[4] Cohn, P.M. (1981): *Universal Algebra*, C. Reidel, Dordrecht, Holland.

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

[6] Delisle, N.M., Menicosy, D.E. and Kerth, N.L. (1982): Tools for supporting structured analysis, *Automated Tools for Information Systems Design*, Schneider, H.-J. and Wasserman, A.I. (eds.), North-Holland, Amsterdam, pp. 11−20.

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

[8] DeMarco, T. and Soceneantu, A. (1984): SYNCRO: a data flow command shell for the Lilith/Modula computer, *Proceedings of 7th International Conference on Software Engineering*, IEEE, New York, pp. 207−213.

[9] Futatsugi, K. and Goguen, J. A. et al. (1985): Principles of OBJ2, *Proceedings of Principles of Programming Symposium*, ACM.

[10] Gane, C. and Sarson, T. (1979): *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Englewood Cliffs, New Jersey.

[11] Goguen, J.A., Meseguer, J. and Plaisted, D. (1983): Programming with parameterized abstract objects in OBJ, *Theory and Practice of Software Technology*, Ferrari, D. and Goguen, J.A. (eds.), North-Holland, Amsterdam, pp. 163−193.

[12] 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.

[13] Goguen, J.A., Thatcher, J.W. and Wagner, E.G. (1978): An initial algebra approach to specification, correctness and implementation of abstract data types, *Data Structuring*, Yeh, R.T. (ed.), *Current Trends in Programming Methodology*, Vol. IV, Prentice-Hall, Englewood Cliffs, New Jersey, pp. 80−149.

[14] Jackson, M.A. (1975): *Principles of Program Design*, Academic Press, London.
[15] Jackson, M.A. (1983): *System Development*, Prentice-Hall, Englewood Cliffs, New Jersey.
[16] Lauber, R.J. (1982): Development support systems, *IEEE Computer*, Vol. 15, No. 5, pp. 36−46.
[17] Orr, K.T. (1977): *Structured Systems Development*, Yourdon, New York.
[18] Sannella, D. (1984): A set-theoretic semantics for Clear, *Acta Informatica*, Vol. 21, pp. 443−472.
[19] Shigo, O., Iwamoto, K. and Fujibayashi, S. (1980): An software design system based on a unified design methodology, *Journal of Information Processing*, Vol. 3, No. 3, pp. 186−196.
[20] Tse, T.H. (1985): ‘‘An automation of Jackson’s structured programming’’, *Australian Computer Journal*, Vol. 17, No. 4.
[21] 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.
[22] Yourdon, E. and Constantine, L.L. (1989): *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*, Prentice-Hall, Englewood Cliffs, New Jersey.