STEPWISE SPECIFICATION AND IMPLEMENTATION
OF ABSTRACT DATA TYPES

Hartmut Ehrig
Hans-Jörg Kreowski
Peter Paßawitz
Technische Universität Berlin, Fachbereich Informatik
März 1978

ABSTRACT

The algebraic approach to specification and implementation of abstract data type in the sense of Goguen, Thatcher and Wagner is extended to study problems of step-wise specification and implementation. Two different concepts are introduced:

1. Stepwise specification by enrichment
2. Stepwise specification and implementation by functors

In both cases some basic results are given and applied to a practical example of software engineering.

INTRODUCTION

The initial algebra approach to the specification, correctness, and implementation of abstract data types in /GTW 76/ seems to be most promising with respect to problems of correct specification and implementation of data structures and software modules. Whereas in /LZ 74/, /Zil 75/, /Gut 75/ and /Gut 76/ the ideas of abstract data types resp. an algebraic approach to these problems is discussed for a few examples the basis for a rigorous mathematical theory was first presented in /ADJ 75/ and a more detailed version in /GTW 76/. Meanwhile algebraic specifications have been studied by several authors, e.g. in /GGM 76/, /Ehr 77/, /Wan 77/, /BG 77/, /EKP 77/, /LS 77/, /TW 78/ and /EKW 78/. In this paper we want to extend the /GTW 76/-approach to cover problems of stepwise specification and implementation. Our concepts and results are applied to a stepwise specification of a symboltable in the sense of /Gut 76/ and to the stepwise specification and implementation of an uncompromisable computer security system in /WOGS 75/.

First we review the basic definitions and constructions for specification of abstract data types given in /GTW 76/. But we give a precise definition for correctness of specifications and state a criterion (Criterion 1.3) for correctness which is only implicit in /GTW 76/.
In Section 2 stepwise specification by enrichment is studied. Enrichment operations - including V-functions in the sense of /Par 72/ - are characterized to be those which are completely and consistently specified (Lemma 2.3). Roughly spoken this corresponds to sufficient completeness and consistency in the appendix of /Gut 76/, but actually our conditions are stronger.

To have simple conditions to be checked in most of the practical examples we introduce inductively specified operations which are shown to be enrichment operations (Theorem 2.13). This is applied to a stepwise correct specification of a symbol-table in the sense of /Gut 76/ (Example 2.14).

While stepwise adjunction of enrichment operations does not change the underlying set of the data structure (resp. the term algebra $T_{\Sigma,E}$ of the specification) stepwise specification by functors allows to enrich not only the set of operations but also the data structure itself. The algebraic connection between two levels of specification $\text{SPEC}_1$ and $\text{SPEC}_2$ is given by a functor which assigns each data structure of Type $\text{SPEC}_1$ one of Type $\text{SPEC}_2$. This concept is implicitly used in the concrete specification of /WOGS 3 75/.

The basic ideas of /WOGS 3 75/ are sketched in this paper where one of the constructions is shown to be a functor (Example 3.2).

But /WOGS 3 75/ is not only an example of stepwise specification but also of stepwise implementation by functors because - as shown in the refined mathematical version /Pad 78/ of /WOGS 3 75/ - a suitable algebraic model of the last specification level can be automatically translated into a SIMULA-implementation of the security system. This was one of the motivations to extend the concept of an implementation based on derivors in /GTW 76/ to our new concept based on functors. Another motivation was to include relevant examples like the implementation of a symbol-table in /Gut 75/ (Example 3.7) which is not covered by /GTW 76/.

On the other hand our concept and also that in /LS 77/ and /TW 2 78/ which are based on arbitrary functors might be too general in view of practical implementations. Probably we need something like an inductively or recursively defined functor to give a general method how to construct implementations like that in /Gut 76/. That seems to be an open problem, perhaps effective categories and functors defined in /Kan 77/ can be used. At least we are able to show that /Gut 76/ and /WOGS 3 75/ are special cases of our general concept based on functors. Moreover an easy consequence of our definition is that implementations are closed under composition (Lemma 3.6), which might be interesting in view of practical applications.

Note that our main examples for stepwise specification in 2.14 and 3.2 can only be sketched in this version for the Proceedings. But the full examples will be included in a version to be published in a journal.