J. Gruska and M. Chytil, editors, Proc. MFCS'81, LNCS 118, pages 271–280, Berlin, 1981. Springer–Verlag

On Realization and Implementation

H.D. Ehrich

Abt.Informatik,Univ. Dortmund, PF 500500,D-4600 Dortmund 50

Abstract - We give a fundamental notion of implementation of one algebraic data type by another one that gives a unifying framework for studying various recent approaches to algebraic implementation, and at the same time is consistent with the classical theory of finite automata realization developed by Hartmanis and Stearns. Much uniformity and simplicity is achieved by discarding reduction problems from automata realization as well as specification problems from algebraic data type implementation. As a result of our approach, we get new insights in the composability of algebraic implementations and the existence of normal forms for algebraic implementations.

1. Introduction

Problems of realization and implementation are of central interest in practical computing, and there is an urgent need for results and methods that help to cope with the severe correctness and reliability problems in this field. The word "realization" is mainly used when speaking about hardware, and in the software area we use the word "implementation". From a fundamental viewpoint, however, there are so many common aspects that it seems reasonable to look for a theoretical foundation for the basic structures and phenomena that occur in both areas.

Hardware realization is treated in the classical theories of automata realization and state assignment. The most general approach that is presented in a mathematically precise way and has found wide acceptance is that of Hartmanis and Stearns (HS 66). So we chose this as a starting point.

On the software implementation side, the choice is not so easy. There is quite a diversity of recent approaches in connection with the theory of abstract data types (ADJ 78, EH 78/79, EKP 78/80, GA 80, HU 80, LI 79, MA 78, NO 79). It is not at all obvious how these approaches are related, although they have a common origin, namely the works of Guttag (GU 75) and ADJ (ADJ 78). We will give some comments on this in section 6, on the basis of our approach developed before in sections 4 and 5.

It is even less obvious how these approaches are related to Hartmanis' and Stearns' theory of automata realization. Although it is standard to view automata as many-sorted algebras (3-sorted , to be specific), automata realization does not easily show up as a special case in any of the approaches to implementation mentioned above. We therefore give a short review of the relevant aspects of automata realization and state assignment in section 3.

The algebraic approach to implementation we have chosen here is a modification of the abstract functional approach in EKP 78. Our modifications, however, remedy some of the problems with that approach. Moreover, we consequently separate problems of (equational) specification from those of implementation. The relationship to automata realization becomes clear by separating this problem in turn from that of automata reduction.

Most of the papers quoted above prefer more concrete approaches to implementation as sequences of certain implementation steps. We model this within our approach by defining derivors, factorizations, restrictions, and declarations as special cases. Our results on composition and normal forms of implementations give some new insights in the composability problems discussed in EH 79 and EKMP 80.

2. FUNDAMENTAL NOTIONS

We shortly review the algebraic concepts and notions needed here. More details can be found in ADJ 77, ADJ 78, and AM 75.

A signature is a quadruple $\Sigma = \langle S, \Omega, arity, sort \rangle$, where S and Ω are sets of sorts and operators, respectively, and $arity: \Omega \to S^*$, $sort: \Omega \to S$ are mappings. We will write $\Sigma = \langle S, \Omega \rangle$ for short, assuming tacitly the existence of the arity and sort mappings. A signature morphism $f: \Sigma \to \Sigma'$ is a pair of mappings $f = \langle f_S: S \to S', f_\omega: \Omega \to \Omega' \rangle$ such that $arity(\omega f_\omega) = arity(\omega) f_S$ and $sort(\omega f_\omega) = sort(\omega) f_S$. For convenience, we often omit the index, writing f for f_S or f_ω.

Algebras are interpretations of signatures: a Σ-algebra A is an S-indexed family of sets, $\{s_A\}$, the carrier of A, together with an Ω-indexed family of mappings, $\{\omega_A: arity(\omega)_A \to sort(\omega)_A\}$, the operations of A (if $x = s_1 s_2 \ldots s_n \in S^*$, x_A denotes the cartesian product $s_{1,A} \times \ldots \times s_{n,A}$). A ($\Sigma$-)subalgebra of A is an S-indexed family of subsets of the s_A that is closed under the operations, together with the restrictions of the operations to these subsets. A Σ-algebra morphism $\varphi: A \to B$ is an S-indexed family of mappings $\varphi_s: s_A \to s_B$ such that, for each operator $\omega \in \Omega$ with arity x and sort s, we have $\omega_A \varphi_s = \varphi_x \omega_B$. Here, $\varphi_x = \varphi_{s_1} \times \ldots \times \varphi_{s_n}$ if $x = s_1 \ldots s_n$. The class of all Σ-algebras with all Σ-algebra morphisms forms a category Σ-alg. I is well known that Σ-alg has an initial algebra I_Σ, having a unique morphism to any other algebra in Σ-alg.

If $f: \Sigma \to \Sigma'$ is a signature morphism, there is a corresponding forgetful functor f-alg : Σ'-alg $\to$ Σ-alg sending each Σ'-algebra B to that Σ-algebra A such that $s_A = (sf)_B$ and $\omega_A = (\omega f)_B$. There is also a functor in the reverse direction, f-free: Σ-alg $\to$ Σ'-alg, sending each Σ-algebra A to the free Σ'-algebra over A. That is, there is a morphism $\eta_A: A \to A.$ f-free. f-alg, the "inclusion of generators",

with the following property: for each morphism $g:A \longrightarrow B.\underline{f\text{-}alg}$, there is a unique morphism $g^{\#}:A.\ \underline{f\text{-}free} \longrightarrow B$ "extending" g, i.e. satisfying $\eta_A \cdot g^{\#}\ \underline{f\text{-}alg} = g$. If f is an inclusion, a standard construction for the free algebra $A.\underline{f\text{-}free}$ over A roughly works as follows: first construct the S'-indexed set of all Ω'-terms over elements of A. Together with the operations of term construction (formal application), this gives a Σ'-algebra A'. Since $\Omega \subset \Omega'$, among all Ω'-terms there are all Ω-terms, and these denote specific values in A. Let $\equiv$ be the congruence generated by all equations of the form t=a that are valid in A, where t is an Ω-term and a is an element of A. Then $A.\underline{f\text{-}free} \cong A'/\equiv$.

3. REALIZATION OF FINITE AUTOMATA

We shortly summarize those aspects of the theory of Hartmanis and Stearns (HS 66) that are of relevance here. Then we give a uniform reformulation of the notion of automata realization that allows for generalization to algebras with arbitrary signatures.

Let two automata be given, $A=(S_A, X_A, Y_A, \delta_A, \lambda_A)$ and $T=(S_T, X_T, Y_T, \delta_T, \lambda_T)$. T is called the target. According to (HS 66), a realization of T by A is a triple α of mappings

$$\alpha_x:X_T \longrightarrow X_A\ , \qquad \alpha_y:Y_A \longrightarrow Y_T\ , \qquad \alpha_s:S_T \longrightarrow \mathcal{P}(S_A)-\emptyset$$

such that (1) $\delta_A(\alpha_s(s), \alpha_x(x)) \subset \alpha_s(\delta_T(s,x))$

and (2) $\alpha_y(\lambda_A(s', \alpha_x(x)))=\lambda_T(s,x)$

for each $s' \in \alpha_s(s)$.

This notion gives an asymmetric treatment to the three sorts and two operators in question, and we would like to look for a uniform characterization that allows for generalization. First, it is well known that two states $s_1, s_2 \in S_T$ are equivalent if their α_s-images are nondisjoint. As a corollary, if T is reduced, then the $\alpha_s(s)$, $s \in S_T$, are pairwise disjoint. One of the central results of Hartmanis and Stearns concerning realization is the following: A realizes T with injective α_x iff there is a subautomaton A' of A and a surjective automata morphism h from A' onto the reduced automaton T_{red} of T. Thus, for reduced targets T and injective α_x, realizations coincide with surjective morphisms from a subautomaton of A to the target. This is nicely uniform, so let us have a closer look on what the restrictions mean.

Now, if α_x is not injective, say $\alpha_x(x)=\alpha_x(x')$ while $x \neq x'$, we easily see that, for the target T, we have $\lambda_T^*(s,\bar{x}x\bar{x}')=\lambda_T^*(s,\bar{x}x'\bar{x}')$ for all $s \in S_T$ and all $\bar{x}, \bar{x}' \in X_T^*$. Especially for each $s \in S_T$, $\delta_T(s,x)$ is equivalent to $\delta_T(s,x')$, i.e. if T is reduced, we have

$\delta_T(s,x) = \delta_T(s,x')$. By extending the notion of equivalence and reduction to inputs in an obvious way, α_X would be automatically injective for any reduced T.

Assuming this general notion of reduction, we have the following situation: a given automaton A realizes a given target T iff, for some A' and some h, we have

$$A \supseteq A' \xrightarrow{h} T_{red} \xleftarrow{g} T$$

Here, g is the unique surjective morphism from T to the reduced automaton T_{red} of T. We see that this notion of realization is in fact a mixture of two notions, represented by the two parts of the above diagram to the left and to the right of T_{red}. The latter is well known as reduction. We propose to take the left one as a (reduction independent) notion of realization, i.e. A realizes T iff there is an A' and an h such that $A \supseteq A' \xrightarrow{h} T$. Realizations in the sense of Hartmanis and Stearns are then recovered by first reducing (including the inputs) and then realizing in this sense.

While we feel it appropriate to do away with reduction when studying realization, we feel on the other hand that the problems of "state assignment" (suitably extended to inputs and outputs) are an integral part of realization. State assignment means to define automaton A in terms of bit vectors and the operations defined on them like shift, negation, conjunction, disjunction, etc. Let the signature of automata be Σ_T. From an algebraic viewpoint, state assignment means to define a Σ_T-algebra structure A on a given algebra B of bit vectors with quite a different signature, say Σ_B. Thus, realization describes the following situation:

$$B \xrightarrow{F} A \supseteq A' \xrightarrow{h} T \quad .$$

We call B the base of the realization. We assume that F represents a uniform way of imposing Σ_T-algebra structures on Σ_B-algebras that can be expressed as a functor $F : \underline{\Sigma_B\text{-alg}} \longrightarrow \underline{\Sigma_T\text{-alg}}$. This very general approach will be restricted and made more concrete in section 5 by considering special cases of functors describing typical methods of defining Σ_T-algebras on Σ_B-algebras.

4. ALGEBRAIC IMPLEMENTATION

Typical examples of the situation to be modelled here are stacks implemented by arrays, symbol tables implemented by stacks of arrays, etc. In the algebraic approach to data types, we have, in each case, a base algebra B of signature Σ_B and a target algebra T of signature Σ_T. Very abstractly, an implementation is given by first defining a Σ_T-algebra A on B and then giving a rule how to interpret A as the target.

Definition 4.1: An (algebraic) implementation of T by B is a functor

$F : \underline{\Sigma_B\text{-alg}} \longrightarrow \underline{\Sigma_T\text{-alg}}$ such that there is a subalgebra $A' \subset BF$ and a surjective morphism $h:A' \twoheadrightarrow T$. h is called an interpretation morphism of the implementation F.

The morphism h interprets in a sense base entities as target entities. Since h need not be injective, multiple representations of target entitites can be covered. For example, arrays that differ only beyond the top pointer represent the same stack. Since h need not be defined on the whole base, there may be base entities without any interpretation. For example, arrays with negative top pointer represent no stack.

This notion of implementation is similar to that in EKP 78. The main difference is that our interpretation morphism need only be defined on a subalgebra of BF, and it is not part of the notion, only some suitable h must exist. This is justified by the fact that any such h must be defined on the minimal subalgebra A_m of A' (or of BF), and all possible h must coincide on A_m (from A_m, there is at most one morphism to any other Σ_T-algebra). Thus, if T is a minimal algebra as suggested by the initial algebra approach to abstract data types (ADJ 78), the morphism $h_m:A_m \longrightarrow T$ is unique, if it exists. Clearly, h_m is surjective in this case.

There is an interesting connection to Mayoh's notion of implementation of functional data types, based on the idea of simulation (MA 78). His functional data types correspond to initial morphisms to minimal algebras (that are surjective). If $i_B:I_{\Sigma_B} \longrightarrow B$, $i_T:I_{\Sigma_T} \longrightarrow T$, $j:I_{\Sigma_T} \longrightarrow I_{\Sigma_B F}$, and $i_A:I_{\Sigma_T} \longrightarrow A_m$ are the respective initial morphisms, then i_A is essentially $j.i_{BF}$, and we have $i_T = j.i_{BF}.h_m$. Thus, j and h_m play the roles of Mayoh's encode and decode mappings, respectively.

Two special cases of implementation are of particular interest. Let base and target signatures Σ_B and Σ_T as well as a functor $F : \underline{\Sigma_B\text{-alg}} \longrightarrow \underline{\Sigma_T\text{-alg}}$ be given. Furthermore, let B be a Σ_B-algebra, and let T be a Σ_T-algebra.

Definition 4.2: F is called a total implementation of T by B iff there is an interpretation morphism defined totally on BF. F is called an isomorphic implementation of T by B iff there is an interpretation morphism that is an isomorphism.

Isomorphic implementations can be characterized equivalently by the existence of an injective morphism $\bar{h} : T \longrightarrow BF$, given by the inverse of h. We will often use this characterization. In the special case of automata realization, isomorphic implementations coincide with isomorphic realizations (HS 66).

It is important that implementations can be composed from single implementation steps. In addition to the above data, let Σ_Z be another signature, let Z be a Σ_Z-algebra, and let G be an implementation of Z by T.

Lemma 4.3 (Composition Lemma):

(1) If G respects injective and surjective morphisms, then FG is an implementation of Z by B.

(2) If F and G are total implementations and G respects surjective morphisms, then FG is a total implementation of Z by B.

(3) If F and G are isomorphic implementations and G respects injective morphisms, then FG is an isomorphic implementation of Z by B.

5. STEPWISE IMPLEMENTATION

It is a practical requirement that implementations should be syntactically describable. Thus, an especially important class of implementations arises from functors of the form f-alg or f-free for a given signature morphism $f: \Sigma_B \longrightarrow \Sigma_T$. As a shorthand notation, let $\bar{F}$=f-alg and F=f-free. Let a basis $B \in \underline{\Sigma_B\text{-alg}}$ and a target $T \in \underline{\Sigma_T\text{-alg}}$ be given.

We now consider four special cases of implementations. The first three have been of central interest in the literature, namely derivors, factorizations, and restrictions (ADJ 78, EH 79, NO 79, LI 79, EKP 80). The last one, called declaration, allows to introduce new sorts and is similar to a concept introduced in EKP 80.

Derivors cover the idea of adding new operations ("procedures") to the base and defining them completely and consistently in terms of the base operations. Factorizations cover the idea of identifying certain entities as representing the same target entity, i.e. forming a quotient structure of the base. Restrictions cover the idea of considering only a part of the base as representing target entities, i.e. forming a subalgebra of a reduct. Declarations cover the idea of adding new data structures to the base and "declaring" them by giving new sorts and construction operators for them.

Definition 5.1: A derivor from B to T is a free functor F of a signature morphism of the form $f: \Sigma_B \hookrightarrow \Sigma_B + \langle \emptyset, \Omega' \rangle$, such that $B \cong T\bar{F}$.

It is easy to see that such an F is a total implementation of T by B. Up to isomorphism, this means that T has the same carrier and operations as B, and in addition the new operations of Ω' are totally defined on the carrier.

Definition 5.2: A factorization from B to T is an identity functor that is a total implementation of T by B.

Thus, for factorizations we have $\Sigma_B = \Sigma_T$, and there is a surjective morphism $h: B \twoheadrightarrow T$, i.e. T is isomorphic to a quotient structure of B.

Definition 5.3: A restriction from B to T is an algebraic functor $\bar{F}$ of a signature morphism of the form $f: \Sigma_T \hookrightarrow \Sigma_T + \langle S', \Omega' \rangle$, such that F is an isomorphic implemen-

tation of T by B.

Up to isomorphism, the injective morphism $\bar{h}:T \rightarrowtail B\bar{F}$ characterizes T as a subalgebra of the reduct $B\bar{F}$ of B.

Definition 5.4: A declaration from B to T is a free functor F of a signature morphism of the form $f:\Sigma_B \hookrightarrow \Sigma_B+\langle S', \Omega'\rangle$ such that (1) F is a total implementation of T by B, (2) $T\bar{F} \cong B$, and (3) sort $(\omega)\in S'$ and arity $(\omega)\notin S'^{+}$ for each operator $\omega\in\Omega'$. F is called primitive iff, for each operator $\omega\in\Omega'$, arity $(\omega)\in S^{+}$ (where S is the sort set of Σ_B).

Thus, up to isomorphism, T consists of B and some new carriers that are in a sense generated by B and the new operations. These are grounded on B, i.e. if the arity is not empty, it contains at least one old sort from S. For primitive operations, the arity must not be empty, and it must contain only old sorts. Examples of primitive declarations are products and sums, whereas the general case allows for stacks, queues, sets, etc. as new data structures.

From the usual term construction of free algebras we see that $B\bar{F}\tilde{F}\cong B$. Since $T\bar{F}\cong B$, there is a (surjective) morphism $h:B\bar{F} \twoheadrightarrow T$ such that $h\bar{F}$ is an isomorphism. This means intuitively that T is (isomorphic to) a quotient of $B\bar{F}$ with respect to a congruence relation that is the identity on the base sorts in S_B.

The interesting point about derivors, factorizations, restrictions and declarations is that they can be freely composed, i.e. their composition in any order and length gives again an implementation. This follows from the following theorem.

Theorem 5.5: Let $F: \underline{\Sigma_B\text{-alg}} \longrightarrow \underline{\Sigma_T\text{-alg}}$ be a functor. If F is of the form $F=F_1F_2$ where F_1 is an implementation of Z by B, and F_2 is a derivor, a factorization, a restriction, or a declaration from Z to T, then F is an implementation of T by B.

Proof: It is straightforward to check for each of the implementation steps in question that it respects injective and surjective morphisms, considering the standard construction of free algebras or reducts, respectively. Now apply lemma 4.3 (1).

The composition problems discussed in EH 79 and EKMP 80 address the question whether compositions of certain sequences of implementation steps of a fixed type can be constructed that are again of this type. The normal form results in the next section partially solve these problems.

6. NORMAL FORMS

The majority of the approaches to algebraic implementations favor a specific sequence of implementation steps as the fundamental notion of implementation. There is, however, no agreement on which sequence that should be, and there is a diversity of

mathematical concepts to express the ideas. In order to give a rough idea, let d,f, r,dcl stand for derivor, factorization, restriction, or declaration, respectively. ADJ 78 favors a 3-step approach of the form d-f-r, where the derivor is restricted to the "nonrecursive" case, i.e. the new operations have to be defined explicitly in terms of the base operations. NO 79 takes the same view in this point. EH 78 and EH 79 favor a 2-step approach where the first step is a combination of d and f, and the second step is r. LI 79 suggests the sequence d-r-f, and a similar approach is taken independently by EKP 80, where the first step, however, is a combination of d and dcl. HU 80 favors a 2-step approach where the first step seems to combine d,dcl and f, and the second step is essentially r. GA 80 considers arbitrary sequences of pairs the first steps of which are essentially combinations of d and f, while the second steps are essentially r.

This situation brings up the question whether there is a fixed sequence of implementation step forms that defines a normal form in the sense that, whenever there is an implementation of T by B composed of d,f,r, and dcl (or some subset thereof), there is an implementation of T by B with this fixed sequence. If this is true, is there a shortest normal form sequence, and which one is it?

These questions can be answered positively from the next lemma that gives a general criterion for the existence of d-r implementations. Let $\Sigma_B=\langle S_B,\Omega_B\rangle$, $\Sigma_T=\langle S_T,\Omega_T\rangle$, and let $S_T\subset S_B$. Let U_B (U_T) be the forgetful functor from $\underline{\Sigma_B\text{-alg}}$ ($\underline{\Sigma_T\text{-alg}}$) to the category of S_B-(S_T-) sorted sets. Let V_{BT} be the forgetful functor from S_B-sorted sets to S_T-sorted sets defined by $v\colon \langle S_T,\emptyset\rangle \hookrightarrow \langle S_B,\emptyset\rangle$ (i.e. $V_{BT}=\underline{v\text{-alg}}$). Let B and T be base and target algebras with signatures Σ_B and Σ_T, respectively. The following lemma is a generalization of theorem 5.5 in EH 79.

Lemma 6.1: There is an implementation $F=F_1\bar{F}_2$ of T by B where F_1 is a derivor and $\bar{F}_2$ is a restriction iff there is an injective S_T-sorted mapping $\bar{h}\colon TU_T \rightarrowtail BU_BV_{BT}$.

As a consequence of this result, we have the following normal form for stepwise implementations without declaration.

Theorem 6.2: If there is an implementation of T by B composed of any sequence of derivors, factorizations, and restrictions, then there is a 2-step implementation of T by B consisting of a derivor and a subsequent restriction.

Proof: Derivors, factorizations, and restrictions can only exist if the cardinality of the base is not less than that of the target, and the same holds for any composition. The theorem now follows from lemma 6.1. If we include declarations, we get the following normal form result.

Theorem 6.3: If there is an implementation of T by B composed of any sequence of derivors, factorizations, restrictions, and (primitive) declarations, then there is a 3-step implementation of T by B composed of a (primitive) declaration, a derivor,

and a restriction - in that order.

This is an easy consequence of the following sharper result.

Theorem 6.4: Any target T can be implemented by any base B by means of a 3-step implementation composed of a declaration, a derivor, and a restriction - in that order. If B is nonempty, the first step can be chosen to be a primitive declaration.

Proof: In the first step, we introduce the target sorts and a suitable set of constructor operators generating carriers with cardinalities not less than those of the target with the same sort. Then we apply the construction of the proof of lemma 6.1.

7. CONCLUSION

We have given a fundamental notion of implementation of one algebra by another one that is consistent with the classical realization theory of finite automata, and provides a unifying framework in which the nature of the various approaches to implementation of abstract data types can be studied and compared. Thus, our approach covers the fundamental aspects of both hardware realization and software implementation showing that these problems have the same abstract structure and can be treated by the same mathematical methods. This uniformity is achieved by discarding reduction problems from automata realization as well as specification problems from those of algebraic implementation.

The practically most relevant cases of implementations studied in this paper are derivors, factorizations, restrictions,and declarations. All of these implementations are syntactically describable by signature morphisms satisfying certain conditions. An important result is that these implementation steps can be composed freely. We have shown, too, that implementations that can be done in terms of these steps can be done - in principle, i.e. without involving effectivity arguments - in certain normal forms consisting of two or three of these steps. Moreover, these implementation steps are powerful enough to implement any given target by any given base.

REFERENCES

ADJ 77 Goguen, J.A./Thatcher,J.W./Wagner,E.G./Wright,J.B.: Initial Algebra Semantics and Continuous Algebras. Journal ACM 24, (1977), 68-95

ADJ 78 Goguen,J.A./Thatcher,J.W./Wagner,E.G.: An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types. Current Trends in Programming Methodology, Vol IV(R.T.Yeh,ed.). Prentice Hall, Englewood Cliffs 1978, 80-149

AM 75 Arbib,M.A./Manes,E.G.: Arrows, Structures, and Functors. Academic Press, New York, 1975

EH 78 Ehrich,H-D.: Extensions and Implementations of Abstract Data Type Specifications. Proc. 7th MFCS 1978, J.Winkowski (ed), LNCS 64, Springer-Verlag, Berlin 1978, 155-164

EH 79 Ehrich,H.-D.: On the Theory of Specification, Implementation, and Parameterization of Abstract Data Types. Bericht Nr. 82/79, Abteilung Informatik, Univ. Dortmund 1979 (also to appear in Journal ACM)

EKMP 80 Ehrig,h./Kreowski,H.-J./Mahr,B./Padawitz,P.: Compound Algebraic Implementations: an Approach to Stepwise Refinement of Software Systems. Proc. 9th MFCS (P.Dembinski, ed.), LNCS 88, Springer-Verlag, Berlin 1980, 231-245

EKP 78 Ehrig,H./Kreowski,H.-J./Padawitz,P.: Stepwise Specification and Implementation of Abstract Data Types. Proc. 5th ICALP (G. Ausiello/C.Boehm, eds.), LNCS 62, Springer-Verlag, Berlin 1978, 205-226

EKP 80 Ehrig,H./Kreowski,H.-J./Padawitz,P.: Algebraic Implementation of Abstract Data Types: Concept, Syntax, Semantics, and Correctness. Proc. 7th ICALP (J.W. deBakker/J. van Leeuwen, eds.), LNCS 85, Springer-Verlag, Berlin 1980, 142-156

GA 80 Ganzinger,H.: Parameterized Specifications: Parameter Passing and Implementation. Internal Report, EECS-Comp.Sc. Division, UC Berkeley, September 1980

GU 75 Guttag, J.V.: The Specification and Application to Programming of Abstract Data Types. Tech. Report CSRG-59, Univ. of Toronto, September 1975

HS 66 Hartmanis,J./Stearns, R.E.: Algebraic Structure Theory of Sequential Machines. Prentice-Hall, Englewood Cliffs, 1966

HU 80 Hupbach,U.L.: Abstract Implementations of Abstract Data Types. Proc 9th MFCS (P.Dembinski, ed.), LNCS 88, Springer-Verlag, Berlin 1980, 291-304

LI 79 Lipeck, U.: Zum Begriff der Implementierung in der Theorie der abstrakten Datentypen. Diplomarbeit, Abteilung Informatik, Univ. Dortmund 1979

MA 78 Mayoh,B.H.: Data Types as Functions. Report DAIMI PB-89, Comp. Sc.Dept. Aarhus University, July 1978

NO 79 Nourani,F.: Constructive Extension and Implementation of Abstract Data Types and Algorithms. Report UCLA-ENG-7945, Comp. Sc. Dept. UC Los Angeles, August 1979