Implementation of Parameterized Observational Specifications

Rolf Hennicker
Fakultät für Mathematik und Informatik
Universität Passau
Postfach 2540
D-8390 Passau

Abstract
An observational approach to the modular construction of algebraic implementations is presented. Based on the theory of parameterized observational specifications an implementation relation is defined which formalizes the intuitive idea that an implementation is correct if it produces correct observable output. It is shown that observational implementations compose vertically and (under appropriate conditions) horizontally. To be useful in practice proof theoretic criteria for parameterized observational implementations are given which are based on the notion of observable parameter context.

1. Introduction

An implementation concept for parameterized algebraic specifications is presented which is based on an observational approach to software development. The basic assumption is that from the software user's point of view a software product is a correct implementation if it satisfies the desired input/output behaviour, independently of the internal properties of a program which may not satisfy a given specification. For example the familiar array-pointer realization of stacks does not satisfy the stack equation pop(push(x, s)) = s or the usual implementation of sets by lists does not satisfy the characteristic set equations. Nevertheless, these implementations are considered to be correct since they produce correct observable output.
In the framework of algebraic specifications this means informally that a concrete specification SP1 is a correct implementation of an abstract specification SP if it preserves the observable properties of SP. To formalize this principle one needs a formal notion of observability which allows to abstract from the internal (non observable) details of data structures. In the literature several behavioural approaches were studied which principally agree in their view of behavioural equivalence of algebras (either with respect to a set of visible (or primitive) sorts (cf. [Giarratana et al. 76], [Reichel 81], [Goguen, Meseguer 82], [Broy et al. 84], [Schoett 87], [Nivela, Orejas 87]) or wrt. a set of observable terms (cf. [Wirsing 86], [Sannella, Tarlecki 87]) or wrt. a set of observable formulas (cf. [Sannella, Tarlecki 85], [Pepper 83])). Our approach is based on observational specifications (cf. [Hennicker, Wirsing 85]) which allow a direct axiomatization of the observable properties of data structures by means of an observability predicate. Having specified a behaviour one can abstract from the model class (of an observational specification) by constructing its behaviour class which is the closure of the model class under observational equivalence. Based on this abstraction principle a simple formalization of the notion of observational implementation is possible:

An observational specification SP1 is an observational implementation of SP if the behaviour class of SP1 (after appropriate restriction) is a subclass of the behaviour class of SP.

Obviously, this implementation relation is transitive, i.e. observational implementations compose *vertically* (cf. [Goguen, Burstall 80]) which is a basic requirement for the construction of correct programs by stepwise refinement.
In practice often large specifications have to be implemented. In this case it should be possible to decompose the (abstract) specification and to implement parts of it independently from each other (e.g. by different programmers). In order to get a correct implementation of the whole system it is necessary that the single "implementation pieces" compose horizontally (cf. [Goguen, Burstall 80]). To support the modular construction of observational implementations we introduce parameterized observational specifications and their implementation.
As usual parameterized observational specifications have a formal parameter specification and a body which in our framework are both observational specifications. Semantically, a parameterized specification is viewed as a function which can be applied to an argument specification and produces a specification as result. The definitions of admissible actual parameter and instantiation are defined similarly to well-known constructions (cf. e.g. [Ganzinger 83], [Ehrig, Mahr 85]). Since parameterized observational specifications

are treated as (partial) functions the observational implementation relation can be extended pointwise to the parameterized case:

A parameterized observational specification P1 is called (parameterized) observational implementation of P (with common formal parameter) if for all admissible parameters SP the application P1(SP) is an observational implementation of P(SP).

This pointwise implementation definition is different from the (proof theoretic) approach of [Ganzinger 83] and the concept of [Ehrig, Kreowski 82] which is based on the initial algebra approach and the free functor semantics. Further implementation concepts for parameterized specifications using a pointwise definition like ours (but based on different implementation notions for (non parameterized) specifications) are provided in [Sannella, Wirsing 82], [Wirsing 86] and [Sannella, Tarlecki 87].

The simplicity of our basic implementation concept results from the fact that observational specifications describe the observable behaviour of data structures and hence determine (on an abstract level) already all possible implementations. On the contrary, implementation concepts adopting the "forget-restrict-identify" method (cf. e.g. [Ehrig et al. 82], [Sannella, Wirsing 82], [Broy et al. 86]) require to connect the models of the implementation with models of the abstract specification e.g. by means of a congruence relation or an abstraction homomorphism.
Related to our notion of implementation are the implementation concepts of [Goguen, Meseguer 82] and [Sannella, Tarlecki 85]. While in [Goguen, Meseguer 82] abstract specification and implementation are persistent extensions of the same specification with all sorts visible, [Sannella, Tarlecki 85] adopt a simple notion of implementation which is based on abstraction with respect to a set of observable formulas. More generally, [Sannella, Tarlecki 87] propose implementations with respect to an arbitrary abstractor. As a main difference to our approach abstractors are determined semantically by an equivalence relation on classes of algebras whereas observational implementations are based on an axiomatization of the observable behaviour and hence are appropriate for the development of proof theoretic implementation criteria.
Based on the notion of observable context a proof theoretic criterium for observational implementations is presented which roughly says that a specification SP1 is an observational implementation of SP if all applications of observable contexts to the axioms of SP yield a sentence which is deducible from the axioms of SP1 (e.g. an observational implementation of the usual specification of stacks (with observable top elements) not necessarily has to satisfy the stack equation pop(push(x, s)) = s but has to satisfy all applications of observable contexts to this equation as e.g. the equation top(pop(push(x, s))) = top(s)). It is shown that this condition can be extended to a context criterium for (parameterized) observational implementations by using observable parameter contexts. In particular the context criteria can be applied to specifications with fixed sets of observable (or visible) sorts (as e.g. in [Goguen, Meseguer 82]).
As already mentioned it is an important issue whether implementations compose horizontally. In our framework this means: If P1 and P are parameterized observational specifications such that P1 implements P and if SP1 and SP are actual parameters such that SP1 implements SP, is the application P1(SP1) an observational implementation of P(SP)?
It is shown that observational implementations compose horizontally if one of the parameterized specifications involved is *monotonic*. For the *monotonicity* of parameterized specifications a sufficient criterium is provided (using the notions of *parameter completeness* and *parameter tolerancy*). The stepwise and modular construction of observational implementations is demonstrated by an example.
The paper is organized as follows: In section 2 the basic notions of observational specifications (cf. [Hennicker, Wirsing 85]) are summarized which are necessary for the following sections. In section 3 the concept of observational implementations is defined and a proof theoretic context criterium is provided. Section 4 introduces parameterized observational specifications and their instantiation and in section 5 the observational implementation relation is extended to the parameterized case. In section 6 the horizontal composition of observational implementations is discussed and illustrated by an example.

2. Basic notions

We assume the reader to be familiar with the basic notions of algebraic specifications (cf. e.g. [Ehrig, Mahr 85]), that are the notions of *signature* $\Sigma = (S, F)$, *signature morphism* σ, *total* Σ*-algebra* $A = ((A_s)_{s\in S}, (f^A)_{f\in F})$, where A_s denotes the carrier sets of A and f^A the total operations of A, *term algebra* $W_\Sigma(X)$ over a S-sorted family $X = (X_s)_{s\in S}$ of sets of identifiers, *ground term algebra* W_Σ, *term* $t \in W_\Sigma(X)$, *ground term* $t \in W_\Sigma$, *substitution* $\sigma: X \to W_\Sigma(X)$, *instantiation* $\sigma(t) = t[\sigma(x_1)/x_1,\ldots, \sigma(x_n)/x_n]$ (i.e. replacement of the

identifiers $x_1,\ldots, x_n \in X$ occurring in t by the terms $\sigma(x_1),\ldots, \sigma(x_n)$), *valuation* $\alpha: X \to A$, *interpretation* of a term t wrt. α, and *finitely generated* (or *term generated*) Σ-algebra.

Moreover, a total Σ-algebra B is called Σ-*subalgebra* of A if $B_s \subseteq A_s$ for all $s \in S$ and $f^A|_B = f^B$ for all function symbols $f \in F$, where $f^A|_B$ denotes the restriction of f^A to the elements of B. For every Σ-algebra A there exists a smallest finitely generated Σ-subalgebra.

A signature $\Sigma' = (S', F')$ is called *subsignature* of Σ if $S' \subseteq S$ and $F' \subseteq F$. The *restriction* of a total Σ-algebra A to Σ' is the Σ'-algebra $A|_{\Sigma'} = ((A_s)_{s\in S'}, (f^A)_{f\in F'})$.

In the following we give a short overview of the theory of observational specifications introduced in [Hennicker, Wirsing 85]. Observational specifications provide an axiomatic description of the observable behaviour of data structures. For that purpose conditional equational specifications are extended by an *observability predicate* "Obs" for specifying the observable objects of data structures.

2.1 Definition An *observational specification* SP is a pair $SP = (\Sigma, E)$ consisting of a signature Σ and a set E of Horn formulas of the form (*)

$$(*) \qquad \phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi_{n+1},$$

where $\phi_1,\ldots, \phi_{n+1}$ are atomic formulas. An atomic formula is either an equation $t_i = r_i$ or an *observation* of the form $Obs(t_i)$ (with terms $t_i, r_i \in W_\Sigma(X)$). The Horn formulas of E are called *axioms* of SP. ◊

The semantics of an observational specification $SP = (\Sigma, E)$ is defined to be the class af all observational Σ-algebras satisfying the axioms of E:

2.2 Definition Let $\Sigma = (S, F)$ be a signature. An *observational Σ-algebra* is a pair (A, Obs^A) consisting of a total Σ-algebra A and a family $Obs^A = (Obs^A_s)_{s\in S}$ of subsets $Obs^A_s \subseteq A_s$.
Obs^A is called *observable part* of A. The elements of Obs^A are called *observable objects.*

An observational Σ-algebra (A, Obs^A) is called *finitely generated* (or *term generated*) if the total Σ-algebra A is finitely generated. An observational Σ-algebra (B, Obs^B) is called *observational Σ-subalgebra* of (A, Obs^A) if the total Σ-algebra B is a Σ-subalgebra of A and if for all sorts $s \in S$: $Obs^B_s = Obs^A_s \cap B_s$.

Let $\Sigma' = (S', F')$ be a subsignature of Σ (i.e. $S' \subseteq S$, $F' \subseteq F$). The *restriction* of (A, Obs^A) to Σ' is the observational Σ'-algebra $(A|_{\Sigma'}, Obs^A|_{\Sigma'})$ where $A|_{\Sigma'}$ is the restriction of A to Σ' (see above) and $Obs^A|_{\Sigma'} = (Obs^A_s)_{s\in S'}$. The finitely generated Σ'-subalgebra of $(A|_{\Sigma'}, Obs^A|_{\Sigma'})$ is denoted by $\langle(A, Obs^A)\rangle_{\Sigma'}$.
The *satisfaction relation* is the classical one of first order predicate calculus, whereby all axioms of a specification are assumed to be universally quantified. In particular, an observation Obs(t) ($t \in W_\Sigma(X)$) is valid in an observational Σ-algebra (A, Obs^A) (written $(A, Obs^A) \models Obs(t)$) iff for all valuations $\alpha: X \to A$ the interpretation of t wrt. α is an observable object of (A, Obs^A). ◊

2.3 Definition A finitely generated observational Σ-algebra (A, Obs^A) is called *model* of an observational specification SP if (A, Obs^A) satisfies all axioms of SP. The *model class* of SP is denoted by Mod(SP). ◊

2.4 Definition Let (A, Obs^A) and (B, Obs^B) be finitely generated observational Σ-algebras.
(A, Obs^A) and (B, Obs^B) are called *observationally equivalent* (written $(A, Obs^A) \sim_{obs} (B, Obs^B)$) iff for all ground terms $t, r \in W_\Sigma$:

$$((A, Obs^A) \models t = r \text{ and } (A, Obs^A) \models Obs(t)) \text{ iff } ((B, Obs^B) \models t = r \text{ and } (B, Obs^B) \models Obs(t)) \qquad ◊$$

Observational equivalence defines an equivalence relation on the class of finitely generated observational Σ-algebras.

2.5 Fact
Model classes of observational specifications are in general not closed under observational equivalence. (For example lists are observational equivalent to sets if (only) the results of the *iselement operation* "$\in$" are specified as observable. But lists are not a model of a specification of sets with the usual set equations as axioms.)

2.6 Definition Let SP = (Σ, E) be an observational specification. The closure of the model class of SP under behavioural equivalence is called *behaviour class* of SP and denoted by Beh(SP), i.e.

$$Beh(SP) = \{(B, Obs^B) \mid (B, Obs^B) \text{ is a finitely generated observational } \Sigma\text{-algebra and there exists a model } (A, Obs^A) \in Mod(SP) \text{ such that: } (A, Obs^A) \sim_{obs} (B, Obs^B)\}.$$

Algebras $(B, Obs^B) \in Beh(SP)$ are called *behaviours* of SP. ◊

The construction of the behaviour class of an observational specification provides a uniform abstraction principle for observational specifications which is the basis for the definition of observational implementations.

3. Observational implementations

An important application domain for algebraic specifications is in the formal development of programs by stepwise refinement, a programming discipline which has been proposed already in the beginning of the seventies by Wirth and Dijkstra. Starting from an abstract problem specification one proceeds by constructing step by step more concrete specifications, whereby each step refines the step before by making some design decisions (e.g. choice of data representations, choice of algorithms, etc.) and by elaborating a more detailed description of the problem. For achieving formally the correctness of the final product of a development process a formal notion of implementation is necessary.
From the observational point of view one obtains intuitively the following simple principle of correct implementation:

A specification SP1 is an observational implementation of a specification SP if SP1 preserves the "observable behaviour" of SP, i.e. all algebras satisfying the observable properties of SP1 satisfy the observable properties specified by SP as well.

This informal notion of implementation can be simply formalized by means of the abstraction principle for observational specifications. For technical simplicity we assume that the signature $\Sigma 1$ of the concrete specification SP1 comprises the signature Σ of the abstract specification SP. (i.e. the construction of the implementing specification SP1 by an appropriate enrichment of a given specification, say SP1', is assumed to be already done). Now the implementation principle from above can be formalized as follows:

An observational specification SP1 is an observational implementation of SP if for any behaviour $(B, Obs^B) \in Beh(SP1)$ some appropriately defined restriction of (B, Obs^B) to an observational Σ-algebra belongs to the behaviour class of SP, i.e.

$$Beh(SP1)|_{\text{"appropriate restriction"}} \subseteq Beh(SP).$$

In the simplest case this restriction is just $\langle (B, Obs^B) \rangle_\Sigma$, that is forgetting the sorts and operation symbols of $\Sigma 1$ not belonging to Σ and then constructing the finitely generated Σ-subalgebra (cf. definition 2.2). These steps are usually called *forget* and *restrict*.
In order to get enough generality we allow a further restriction of the observable part Obs^B to a subset $B_0 \subseteq Obs^B$. Informally, the restriction of the observable part means that the implementing specification may specify more objects observable than required by the abstract specification. This coincides with our intuition that implementations fix more and more details whereby the degree of possible abstraction may be limited.
In summary, we get the following definition of observational implementation:

3.1 Definition Let SP1 = (Σ1, E1) and SP = (Σ, E) be observational specifications with $\Sigma \subseteq \Sigma 1$. Let S1 be the set of sorts of Σ1.
SP1 is called *observational implementation* of SP (written SP1 <~~~ SP) if for all behaviours $(B, Obs^B) \in Beh(SP1)$ there exists a family $B_0 = ((B_0)_s)_{s \in S1}$ of subsets $(B_0)_s \subseteq Obs^B{}_s$ such that:

$$\langle (B, B_0) \rangle_\Sigma \in Beh(SP).$$

($\langle (B, B_0) \rangle_\Sigma$ denotes the finitely generated Σ-subalgebra of the restriction of (B, B_0) to Σ (cf. definition 2.2)) ◊

If one wishes to rule out trivial implementations one could simply restrict the class of admissible models (and behaviours) to those algebras satisfying true ≠ false and require consistency of the implementing specification. Under this requirements the criteria for observational implementations (given below) remain valid if the abstract specification SP and the concrete specification SP1 are assumed to contain the basic type BOOL with observable truth values.

The next lemma is an immediate consequence of definition 3.1, definition 2.4 and the transitivity of observational equivalence:

3.2 Lemma SP1 is an observational implementation of SP iff
for all models $(B, Obs^B) \in Mod(SP1)$ there exists a model $(A, Obs^A) \in Mod(SP)$ and a family of subsets $B_0 \subseteq Obs^B$ such that for all ground terms $t, r \in W_\Sigma$:

$$((B, B_0) \models t = r \text{ and } (B, B_0) \models Obs(t)) \text{ iff } ((A, Obs^A) \models t = r \text{ and } (A, Obs^A) \models Obs(t)). \quad \Diamond$$

To be useful in software development by stepwise refinement it is necessary that the composition of consecutive implementation steps yields a single correct implementation step (i.e. composes *vertically* in the sense of [Goguen, Burstall 80]). It is a direct consequence of the definition that observational implementations satisfy this basic requirement:

3.3 Fact The observational implementation relation is transitive.

The notion of observational implementation differs essentially from concepts adopting the "forget-restrict-identify" approach (cf. e.g. [Ehrig et al. 82], [Sannella, Wirsing 82], [Broy et al. 86]). Although the forget-restrict steps correspond to the restriction of behaviours to the signature of the abstract specification the main step in those concepts is the identification of concrete objects which represent the same abstract objects (e.g. by an abstraction function or a congruence relation). Since observational specifications provide a more abstract view of (the semantics of) algebraic specifications this identification can be simply omitted.
Related to our notion of implementation are the implementation concepts of [Goguen, Meseguer 82] and [Sannella, Tarlecki 85]. While in [Goguen, Meseguer 82] abstract specification and implementation are persistent extensions of the same specification with all sorts visible, [Sannella, Tarlecki 85] adopt a simple notion of implementation which is based on abstraction with respect to a set of observable formulas. Compared with [Sannella, Tarlecki 85] (and more generally with [Sannella, Tarlecki 87]) observational implementations correspond to the implementation of an abstract specification after having applied an appropriate abstractor. In our framework no manipulation of the original abstract specification is necessary since observational specifications give already an axiomatization of a behaviour. This axiomatic basis leads to proof theoretic criteria for implementation relations.

An important issue for the application of formal implementation notions in practice is the question for appropriate proof methods (for implementation relations). Since it is highly desirable that correctness proofs are supported by machine we are particularly interested in proof theoretic conditions for implementation relations.
Observational specifications give an axiomatization of the observable behaviour of data structures and hence are appropriate for the development of proof theoretic criteria for implementations. As proof system for observational specifications we generalize the proof system of [Selman 72] to conditional formulas of the form (*) (see definition 2.1) and then extend it by the rule

$$\frac{\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow t = r, \quad \phi_1 \wedge \ldots \wedge \phi_n \Rightarrow Obs(t)}{\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow Obs(r)}$$

which asserts the compatibility of the observability predicate with equality. If a Horn formula $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi_{n+1}$ is deducible from a set of axioms E we write $E \vdash \phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi_{n+1}$. As a simple generalization of theorem 2 in [Selman 72] it can be shown that the proof system "$\vdash$" is sound and complete (wrt. the model class of an observational specification).
For the development of a proof theoretic criterium for observational implementation relations we need the following definitions:

3.4 Definition Let SP = (Σ, E) be an observational specification, let S be the set of sorts of Σ, and let $Z = \{z_s \mid s \in S\}$ be an S-sorted set of identifiers.

1.) A term $c \in W_\Sigma(Z)$ is called *context* over Σ, if c contains exactly one identifier $z_s \in Z$. To indicate the identifier occurring in c we often write $c[z_s]$ instead of c.
The application of a context $c[z_s]$ to a term $t \in W_\Sigma$ of sort s is defined by the substitution of z_s by t. Instead of $c[t/z_s]$ we write briefly c[t].

2.) A sort $s \in S$ is called *observable sort* of SP if there exists an axiom $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow Obs(t)$ of SP such that t is of sort s.

3.) A context $c \in W_\Sigma(Z)$ is called *observable context* of SP if the sort of c is an observable sort of SP. ◊

Using the notion of observable context one can show that an observational specification SP1 implements an observational specification SP (with observable premises in the axioms) if SP1 preserves observability of objects and if SP1 satisfies all Horn formulas of the form $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow c[t] = c[r]$ where $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow t = r$ is (a ground instance of) an axiom of SP and c is an observable context of SP. In particular, SP1 not necessarily satisfies all equational axioms t = r of SP but all applications of observable contexts to (ground instances of) t = r. For example a usual stack specification where the top elements of stacks are specified as observable may be implemented by a specification SP1 which does not satisfy the equation pop(push(x, s)) = s but satisfies all applications of observable contexts to this equation as e.g. the equation top(pop(push(x, s))) = top(s).
Formally, we obtain the following criterium for observational implementations:

3.5 Proposition Let SP1 = (Σ1, E1) and SP = (Σ, E) be observational specifications with $\Sigma \subseteq \Sigma 1$ and let all premises of the axioms of SP be of observable sort (i.e. if p = q is a premises of an axiom of SP then p and q are of observable sort of SP).

SP1 is an observational implementation of SP if the following conditions are satisfied:

a) If $(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow Obs(t)) \in E \ (n \geq 0)$ then $E1 \vdash \sigma^*(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow Obs(t))$
for all (ground) substitutions $\sigma: X \rightarrow W_\Sigma$.

b) For all observable contexts $c[z_s]$ of SP holds:
If $(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow t = r) \in E \ (n \geq 0)$ and if t is of sort s
then $E1 \vdash \sigma^*(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow c[t] = c[r])$ for all (ground) substitutions $\sigma: X \rightarrow W_\Sigma$.

(σ^* denotes the instantiation of formulas wrt. σ. The proof of proposition 3.5 is given in [Hennicker 88], pp. 143.) ◊

Proposition 3.5 gives a practically applicable criterium for implementation relations. For the verification of condition b) in examples the proof technique of *context induction* is appropriate (cf. [Hennicker 88]).

4. Parameterized observational specifications

Parameterized specifications provide a flexible tool for achieving generality and reusability in the process of software specification and development. Similar to a function declaration in some ordinary programming language parameterized specifications have formal parameters and a body which defines the effect of the application to an actual parameter. Actual parameters are just specifications and the result of an application yields again a specification (the instantiation of the body specification). There are several parameterization concepts for algebraic specifications in the literature to some of which it will be referred later in comparison with our approach.
In the following parameterized specifications are studied from the observational point of view. In particular the notion of observational implementation is extended to parameterized specifications.

4.1 Definition A *parameterized observational specification* P is a pair P = (PA, B) consisting of

— an observational specification $PA = (\Sigma_{PA}, E_{PA})$ (called *formal parameter*) and

— an observational specification $B = (\Sigma_B, E_B)$ (called *body*)

such that $\Sigma_{PA} \subseteq \Sigma_B$ and $E_{PA} \subseteq E_B$. ◊

4.2 Example The parameterized observational specification SET describes properties of finite sets which are parameterized with respect to their elements. The formal parameter specification ELEM requires observability of the elements (of sets) and of the boolean values. In particular the results of the *iselem* operation which tests the membership of an element in a set are observable. ELEM contains a subspecification BOOL with the usual laws of the Boolean algebra as axioms (x or x = x, x or y = y or x, etc.), an equality test *eq* for elements which is specified by the axioms of an equivalence relation, and a constant *const*.

Formal parameter:

spec ELEM = **enrich** BOOL **by**
  sorts: elem
  functs: const: → elem
    eq: elem x elem → bool
  axioms:
    $Obs(x_{bool})$, $Obs(x_{elem})$,
    eq(x, x) = true,
    eq(x, y) = eq(y, x),
    eq(x, y) = true ∧ eq(y, z) = true ⇒ eq(x, z) = true

(x_{bool} and x_{elem} are identifiers of sort *bool*, resp. *elem*)

Body:

spec SET (**parameter** ELEM) = **enrich** ELEM **by**
  sorts: set
  functs: empty: → set
    add: elem x set → set
    iselem: elem x set → bool
  axioms:
    iselem(x, empty) = false,
    iselem(x, add(y, s)) = eq(x, y) or iselem(x, s),
    add(x, add(x, s)) = add(x, s),
    add(x, add(y, s)) = add(y, add(x, s)).

◊

Semantically, a parameterized observational specification is considered as a (partial) function which takes an observational specification as argument and yields an observational specification as result. This view coincides with [Sannella, Tarlecki 87] and is basically related to the theory procedures in CLEAR (cf. [Burstall, Goguen 80]) or the λ-calculus like approach of ASL (cf. [Wirsing 86]).
The application of a parameterized observational specification is defined for all actual parameters satisfying the requirements of the formal parameter. Thereby actual parameters are connected to a formal parameter via a signature morphism (from the formal parameter signature to the actual parameter signature) and an actual parameter is called *admissible* if all its models satisfy the (renamed) axioms of the formal parameter specification. Since models are assumed to be term generated this means that all instantiations of the axioms of the formal parameter with ground terms over the signature of the actual parameter are deducible from the axioms of the actual parameter:

4.3 Definition Let P = (PA, B) be a parameterized observational specification with formal parameter PA = (Σ_{PA}, E_{PA}) and body B = (Σ_B, E_B). Let SP = (Σ, E) be an observational specification such that $(\Sigma_B \setminus \Sigma_{PA}) \cap \Sigma = \emptyset$ (i.e. no name clashes).
SP is called *admissible actual parameter* of P with respect to a signature morphism $\rho: \Sigma_{PA} \to \Sigma$ if for all axioms $(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi_{n+1}) \in E_{PA}$ holds:

$$E \vdash \sigma^*(\rho^*(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi_{n+1})) \text{ for all (ground) substitutions } \sigma: X \to W_\Sigma .$$

(ρ^* denotes the extension of ρ to formulas (over Σ_{PA}) and σ^* denotes the instantiation of formulas wrt. σ.) ◊

As an equivalent definition one could require that all (renamed) axioms of PA are in the inductive theory of SP, i.e. are deducible (from the axioms of SP) by infinite induction. This definition of admissible parameter is more liberal than other notions (cf. e.g. [Ehrig, Mahr 85], [Goguen, Meseguer 82]) which require that all axioms of the formal parameter (not only ground instantiations) are provable from the actual ones. Equivalent to our definition is the notion of [Ganzinger 83] (if it is restricted to non parameterized actual parameters). For the CLEAR procedures the more liberal notion of actual parameter can be achieved by using "fitting" morphisms between data theories.
The application of a parameterized observational specification to an actual parameter (also called *parameter passing*) is defined as follows:

4.4 Definition Let P = (PA, B) be a parameterized observational specification with formal parameter PA = (Σ_{PA}, E_{PA}) and body B = (Σ_B, E_B). Let SP = (Σ, E) be an admissible parameter of SP wrt. a signature morphism $\rho: \Sigma_{PA} \to \Sigma$.

The *application* of P to SP wrt. ρ is the observational specification

$$P_\rho(SP) = B[SP/PA]_\rho$$

which is defined by replacing the formal parameter PA by the actual parameter SP while all sorts and function symbols of Σ_{PA} occurring in $B \setminus PA$ are renamed wrt. to ρ. (To simplify the notation we often omit the index ρ and write simply P(SP) instead of $P_\rho(SP)$.) ◊

This syntactic definition of instantiation corresponds to the instantiation concept in [Ehrig, Mahr 85] which is equivalent to a pushout construction (as e.g. in [Goguen, Meseguer 82]).
The application P(SP) of a parameterized observational specification P is called *parameter protecting* if the properties of the actual parameter SP are preserved, i.e. no additional elements of parameter sort are generated by the body ("no junk") and the application does neither introduce new identities for objects of parameter sort ("no confusion") nor additional observable objects (of parameter sort). A necessary and sufficient condition for a parameterized observational specification P to be parameter protecting for all actual parameters is the *parameter completeness* and *parameter tolerancy* of P:

4.5 Definition Let P = (PA, B) be a parameterized observational specification with formal parameter PA = (Σ_{PA}, E_{PA}) and body B = (Σ_B, E_B). Let X_{PA} be a countably infinite set of identifiers of parameter sorts (i.e. sorts of PA).

1.) P is called *parameter complete* if for all terms $t \in W_{\Sigma_B}(X_{PA})$ of parameter sort s there exists a term $p \in W_{\Sigma_{PA}}(X_{PA})$ of sort s such that: $E_B \setminus E_{PA} \vdash t = p$.

2.) P is called *parameter tolerant* if for all observational Σ_{PA}-algebras (A, Obs^A) satisfying the axioms E_{PA} there exists an observational Σ_B-algebra (B, Obs^B) such that (A, Obs^A) is an observational Σ_{PA}-subalgebra of the restriction of (B, Obs^B) to Σ_{PA}. ◊

Parameter completeness together with parameter tolerancy corresponds to the various notions of *persistency* which can be found in the literature. It guarantees that for any actual parameter SP = (Σ, E) of P and for any model (A, Obs^A) of SP there exists an extension to a model $(B, Obs^B) \in Mod(P(SP))$ such that $(B, Obs^B)|_\Sigma = (A, Obs^A)$, i.e. SP is protected by the application.

4.6 Remark Parameter completeness is a necessary condition for achieving that for all actual parameters SP the application P(SP) is sufficiently complete (wrt. ground terms of parameter sort) over SP. Consider for example the specification PA' which is obtained from the formal parameter PA by adding a constant for each parameter sort and by replacing the axioms of PA by all their ground instantiations. Then PA' is an admissible parameter of P but P(PA') is not sufficiently complete over PA' if P is not parameter complete.
The condition of parameter completeness could be weakend to the condition "$E_B \vdash t = p$" if either a less liberal notion of admissible parameter is adopted (see above) or if the instantiation of an actual parameter would be defined by adding the (renamed) axioms of the formal parameter to the actual application (as in [Ganzinger 83]).

5. Implementation of parameterized observational specifications

In this section the notion of observational implementation is extended to parameterized observational specifications. If P1 and P are parameterized observational specifications with common formal parameter then P1 is called *(parameterized) observational implementation* of P if for all admissible parameters SP the application P1(SP) is an observational implementation of P(SP). (For a discussion of different formal parameters see [Sannella, Wirsing 82].) This pointwise definition corresponds exactly to our view of parameterized specifications as functions. Analogous definitions based on different implementation notions (forget-restrict-identify, model class inclusion, abstractor and constructor implementations) are given in [Sannella, Wirsing 82], [Wirsing 86], resp. [Sannella, Tarlecki 87]. A purely proof theoretic implementation concept for parameterized specifications is provided in [Ganzinger 83]. As a major step towards the applicability of algebraic specifications to the modular construction of software systems in [Ehrig, Kreowski 82] it is shown that parameterized implementations based on the initial algebra approach and the free functor semantics are compatible with parameter passing.

5.1 Definition Let P = (PA, B) and P1 = (PA, B1) be parameterized observational specifications. P1 is called *(parameterized) observational implementation* of P (written P1 <~~~ P) if for all admissible parameters SP (wrt. a signature morphism ρ) holds:

$P1_\rho(SP)$ is an observational implementation of $P_\rho(SP)$. ◊

Obviously, the transitivity of the implementation relation for the non parameterized case extends to the parameterized case:

5.2 Fact
The observational implementation relation for parameterized observational specifications is transitive.

As in the non parameterized case for practical applications one is particularly interested in proof theoretic criteria for (parameterized) implementation relations. Proposition 3.5 implies that a parameterized specification P1 is an observational implementation of P if the conditions a) and b) of the proposition can be proved for all applications P1(SP) and P(SP) to actual parameters SP. Hence for the verification all actual parameters have to be considered.
To get rid of reasoning over all actual parameters we are interested in context conditions which only depend on the properties of P and P1 and can be proved independently from actual applications. The basic idea is that an instantiation may impose the observability of *all* parameter sorts and hence we consider (instead of observable contexts (cf. definition 3.4)) all contexts which are either of observable sort *or* of parameter sort. Moreover, since instantiations may introduce arbitrary many ground terms of parameter sort we have to consider contexts which contain arbitrary identifiers of parameter sort.

5.3 Definition Let P = (PA, B) be a parameterized observational specification with formal parameter $PA = (\Sigma_{PA}, E_{PA})$ and body $B = (\Sigma_B, E_B)$ and let S_B be the set of sorts of Σ_B.
Moreover, let $Z = \{z_s \mid s \in S_B\}$ be an S_B-sorted set of identifiers and let X_{PA} be a set of identifiers of parameter sort (as in defintion 4.5) such that Z and X_{PA} are disjoint.

A term $c \in W_{\Sigma_B}(Z \cup X_{PA})$ is called *parameter context* if c contains exactly one identifier $z_s \in Z$ (and arbitrary identifiers of X_{PA}). If the sort of c is either a parameter sort or an observable sort (cf. definition 3.4) of the body B then c is called *observable parameter context* of P. (Context application is defined analogously to definition 3.4 and the notation $c[z_s]$ is used to indicate the identifier z_s of c.) ◊

Using this definition the conditions of proposition 3.5 can be extended to a criterium for parameterized implementation relations. As a prerequisite we require that the implementing parameterized specification is parameter complete. (This could be omitted if we had chosen the alternative definition of parameter passing where the (renamed) axioms of the formal parameter are added to the axioms of the actual application.)

5.4 Proposition Let P1 = (PA, B1) and P = (PA, B) be parameterized observational specifications with bodies B1 = (Σ_{B1}, E_{B1}) and B = (Σ_B, E_B) such that $\Sigma_B \subseteq \Sigma_{B1}$ and let all premises of the axioms of B be of observable sort. Moreover, let P1 be parameter complete.

P1 is a (parameterized) observational implementation of P if the following conditions are satisfied:

a) If $(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \mathrm{Obs}(t)) \in E_B$ $(n \geq 0)$ then $E_{B1} \vdash \sigma^*(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \mathrm{Obs}(t))$
for all substitutions $\sigma: X \rightarrow W_{\Sigma_B}(X_{PA})$.

b) For all observable parameter contexts $c[z_s]$ of P holds:
If $(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow t = r) \in E_B$ $(n \geq 0)$ and if t is of sort s
then $E_{B1} \vdash \sigma^*(\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow c[t] = c[r])$ for all substitutions $\sigma: X \rightarrow W_{\Sigma_B}(X_{PA})$.

(As before σ^* denotes the instantiation of formulas wrt. σ. The proof of proposition 5.4 is given in [Hennicker 88], pp. 117.) ◊

5.5 Example In this example (parameterized) sets are implemented by (parameterized) lists which in turn are implemented by (parameterized) arrays with pointers. Thus, by the vertical composition property (cf. fact 5.2) this yields an observational implementation of sets by arrays with pointers.
In the first step the parameterized specification SET of example 4.2 is implemented by the following parameterized specification LIST which specifies lists in a usual way. Since all elements are specified as observable by the formal parameter ELEM in particular the first elements of lists can be observed by the operation *first*.

spec LIST (**parameter** ELEM) = **enrich** ELEM **by**

sorts: list

functs: empty: → list
add: elem x list → list
first: list → elem
rest: list → list
iselem: elem x list → bool

axioms: first(empty) = const,
rest(empty) = empty,
first(add(x, s)) = x,
rest(add(x, s)) = s,
iselem(x, empty) = false,
iselem(x, add(y, s)) = eq(x, y) or iselem(x, s).

Fact LIST is a (parameterized) observational implementation of SET.

Informally, this fact is clear since lists and sets have the same behaviour wrt. the observable results of the SET operation *iselem*. For a formal proof proposition 5.4 can be applied. The validity of condition b) can be shown by induction on the structure of the parameter contexts. (A detailed proof using the principle of *context induction* is given in [Hennicker 88], pp. 155.)
Note that the sort *set* has to be identified here with the sort *list*. An explicit renaming could simply be treated as in [Sannella, Tarlecki 87].

In the next step lists are implemented by arrays with pointers. The following specification ARRAY_POINTER specifies lists as pairs consisting of an array a (which is parameterized with respect to its entries) and a natural number p (called pointer). The empty list is implemented by the empty array *vac* together with the pointer 0. Adding an element x to a list is implemented by putting x into the (p+1)-th component of the array (if the pointer has value p) and by incrementing the pointer. The *rest* operation is simply implemented by decrementing the pointer (without deleting the last entry) and the first element of a list is obtained by accessing the p-th component of the array (if the pointer has value p).
For simplicity only those array operations are specified here which are necessary for the implementation (*vac* for the empty array, *put* for putting a new element on the array and *get* for selecting an element of the array).

spec ARRAY_POINTER (**parameter** ELEM) =
enrich ELEM, NAT **by**
sorts: array, list
functs: vac: → array
put: array x nat x elem → array
get: nat x array → elem
pair: array x nat → list
empty: → list
add: elem x list → list
first: list → elem
rest: list → list
iselem: elem x list → bool

axioms:
get(p, vac) = const,
get(p, put(a, p, x) = x,
eq_nat(p, q) = false ⇒ get(p, put(a, q, x)) = get(p,a),
empty = pair(vac, 0),
add(x, pair(a, p)) = pair(put(a, p+1, x), p+1),
first(pair(a, p)) = get(p, a),
rest(pair(a, p)) = pair(a, p–1),
iselem(x, pair(a, 0)) = false,
iselem(x, pair(a, p+1)) =
= eq(x, get(p+1, a)) or iselem(x, pair(a, p)).

Note that ARRAY_POINTER does not satisfy the LIST-axiom rest(add(x, s)) = s since the *rest* operation only decrements the pointer and does not delete the element x in the array. But the observable behaviour of LIST is preserved by ARRAY_POINTER since ARRAY_POINTER satisfies all equations between terms (containing only identifiers of parameter sort) of the observable sorts *elem* or *bool* which are satisfied by LIST. In particular ARRAY_POINTER satisfies the equations first(rest(add(x, t))) = first(t) and first(add(x, t)) = x with an identifier x of sort *elem* and a term t with identifiers of parameter sort.

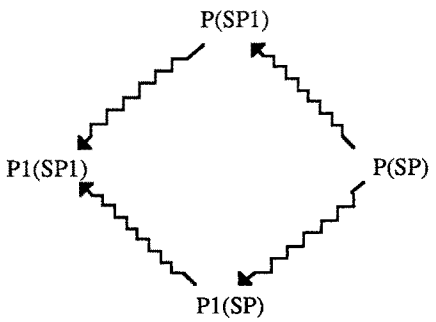
Fact ARRAY_POINTER is a (parameterized) observational implementation of LIST.
(A formal proof by context induction is given in [Hennicker 88], pp. 138.)

By vertical composition of the two implementation steps one obtains

Fact ARRAY_POINTER is a (parameterized) observational implementation of SET. ◊

6. Horizontal composition of observational implementations

It is one of the main issues for the use of formal implementation notions in practice whether they support the modular construction of implementations. More precisely this means that local implementations of parts of a (structured) specification should compose to an implementation of the whole specification. This property is called *horizontal composition* (cf. [Goguen, Burstall 80]). In the framework of parameterized observational specifications the situation can be described as follows:
Given two parameterized observational specifications P1 and P such that P1 <~~~ P and two admissible parameters SP1 and SP such that SP1 <~~~ SP, the question is whether the application P1(SP1) is an observational implementation of P(SP). To study this issue we distinguish two cases for the construction of the implementation P1(SP1) which can be illustrated by the following diagram:



Corresponding to the upper part of the diagram at first the actual parameter SP is replaced locally by its implementation SP1. The resulting specification P(SP1) is an observational implementation of P(SP) if P preserves implementation relations for arguments, i.e. if P is *monotonic*. In a next step P(SP1) is implemented by P1(SP1) which is obviously a correct implementation since it is assumed that P1 implements P.

In the second case corresponding to the lower part of the diagram first P(SP) is implemented by P1(SP) and then P1(SP) is implemented locally by replacing SP by SP1. In this case the resulting specification P1(SP1) is an observational implementation of P1(SP) (and hence of P(SP)) if P1 is *monotonic*.
This discussion shows that the crucial point for achieving horizontal composition is the monotonicity of one of the parameterized specifications involved.

6.1 Definition Let P = (PA, B) be a parameterized observational specification. P is called *monotonic* if for all admissible parameters SP1 and SP which are related by the implementation relation SP1 <~~~ SP holds: P(SP1) <~~~ P(SP).
(More precisely it is assumed that the signature morphisms connecting the formal parameter PA with the actual parameters SP and SP1 coincide.) ◊

We now obtain the following two conditions for the horizontal composition of observational implementations (corresponding to the upper resp. lower part of the above implementation diagram).

6.2 Proposition Let P1 = (PA, B1) and P = (PA, B) be parameterized observational specifications such that P1 <~~~ P. Moreover, let SP and SP1 be admissible parameters such that SP1 <~~~ SP.

1.) If P is monotonic then P1(SP1) <~~~ P(SP).

2.) If P1 is monotonic then P1(SP1) <~~~ P(SP). ◊

Monotonicity is sufficient for the horizontal composition of observational implementations. Compared with the horizontal composition of abstractor implementations in [Sannella, Tarlecki 87] the requirement of monotonicity corresponds to the preservation of abstraction equivalences by the parameterized specifications. Based on a module oriented approach horizontal compositions are studied in [Schoett 87]. The basic requirement there is the stability of the "cells" involved in the program development process. Since our approach admits proof theoretic properties (like parameter completeness) we can give the following sufficient condition for monotonicity:

6.3 Proposition Let P be a parameter complete and parameter tolerant parameterized observational specification. Moreover, let all observable sorts of P be parameter sorts. Then P is monotonic. ◊

(Proposition 6.3 can be proved by a simple generalization of the proof in [Hennicker 88], pp. 153.)

6.4 Example In the following the implementation of sets by arrays with pointers is horizontally composed with an implementation of the integers by sequences of bits. The composition yields an observational implementation of sets of integers by pairs consisting of an array over bit sequences and a pointer.
We start with the specification INT of the intergers. INT specifies the boolean values and all integers (which are constructed by *zero*, *succ*, and *pred*) as observable. For the definition of the equality test *eq_int* the auxiliary function *non_negative* is used. Note that *eq_int* is completely specified for all integers since the equality test can always be reduced to the equality test for non negative integers.

spec INT = **enrich** BOOL **by**
sorts: int
functs: zero: → int
succ, pred: int → int
eq_int: int x int → bool
non_negative: int → bool
axioms:
Obs(true), Obs(false),
Obs(i) ⇒ Obs(succ(i)),
Obs(i) ⇒ Obs(pred(i)),
pred(succ(i)) = i,
succ(pred(i) = i,
eq_int(zero, zero) = true,
eq_int(i, j) = eq_int(j, i),
eq_int(succ(i), succ(j)) = eq_int(i, j),
non_negative(i) = true ⇒ eq_int(zero, succ(i)) = false,
non_negative(zero) = true,
non_negative(pred(zero)) = false,
non_negative(i) = true ⇒ non_negative(succ(i)) = true,
non_negative(i) = false ⇒ non_negative(pred(i)) = false.

Next we give an implementation of integers by pairs $\langle x, s\rangle$ consisting of a sign $x \in \{O, L\}$ and a bit sequence over $\{O, L\}$ (constructed by the constant ε and the operation &). The sign x represents the sign of an integer and the bit sequence s is the usual binary representation of the absolute value. Sequences with leading O and the pairs $\langle O, \varepsilon\rangle$, $\langle L, \varepsilon\rangle$ (representing *zero*) are identified.
For the implementation of *succ* and *pred* the auxiliary function *compl* is used which complements the sign of a bit sequence. That way the computation of the successor (predecessor) of a negative integer can be reduced to the predecessor (successor) of its complement. The equality test *eq_int* is implemented by means of the equality test *eq_seqbit* for bit sequences (taking into account that sequences with leading O are identified). We hope that the basic idea of the implementation is sufficiently illustrated and drop a detailed discussion of the axioms.

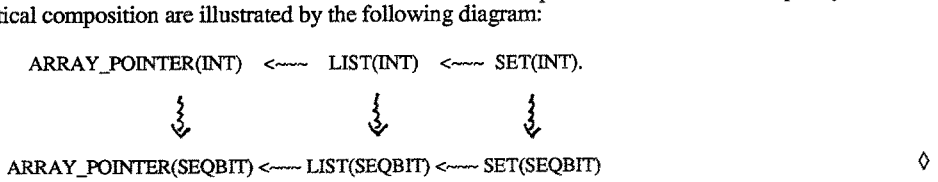
spec SEQBIT = **enrich** BOOL **by**
sorts: bit, seqbit, int
functs: O, L: → bit
eq_bit: bit x bit → bool
ε: → seqbit
. & .: seqbit x bit → seqbit
eq_seqbit: seqbit x seqbit → bool
⟨.,.⟩: bit x seqbit → int
zero: → int
succ, pred, compl: int → int
eq_int: int x int → bool
non_negative: int → bool
axioms:

Obs(true), Obs(false),
Obs(i) ⇒ Obs(succ(i)),
Obs(i) ⇒ Obs(pred(i)),
eq_bit(x, x) = true,
eq_bit(O, L) = false,
eq_bit(L, O) = false,
ε&O = ε,
eq_seqbit(ε, ε) = true,
eq_seqbit(ε, s&L) = false,
eq_seqbit(ε, s&O) = eq_seqbit(ε, s),
eq_seqbit(s&x, s´&y) =
= eq_bit(x, y) and eq_seqbit(s, s´),
eq_seqbit(s, s´) = eq_seqbit(s´, s),
⟨O, ε⟩ = ⟨L, ε⟩,
compl⟨L, s⟩ = ⟨O, s⟩,
compl⟨O, s⟩ = ⟨L, s⟩,
zero = ⟨L, ε⟩,
succ⟨L, ε⟩ = ⟨L, ε&L⟩,
succ⟨L, s&O⟩ = ⟨L, s&L⟩,
succ⟨L, s⟩ = ⟨L, s´⟩ ⇒ succ⟨L, s&L⟩ = ⟨L, s´&O⟩,
succ⟨O, s&x⟩ = compl(pred⟨L, s&x⟩),
pred⟨L, ε⟩ = ⟨O, ε&L⟩,
pred⟨L, s&L⟩ = ⟨L, s&O⟩,
pred⟨L, s⟩ = ⟨L, s´⟩ ⇒ pred⟨L, s&O⟩ = ⟨L, s´&L⟩,
pred⟨O, s&x⟩ = compl(succ⟨L, s&x⟩),
eq_int(⟨x, s⟩, ⟨x, s´⟩) = eq_seqbit(s, s´),
eq_int(⟨O, s⟩, ⟨L, s´⟩) = eq-seqbit(s, ε) and eq-seqbit(s´, ε),
eq_int(i, j) = eq_int(j, i),
non_negative⟨L, s⟩ = true,
non_negative⟨O, s⟩ = eq_seqbit(s, ε).

It can be shown that SEQBIT <~~~ INT and that INT and SEQBIT are admissible parameters of the parameterized specifications SET, LIST, and ARRAY_POINTER (wrt. the signature morphism ρ with ρ(*elem*) = *int*, ρ(*const*) = *zero*, ρ(*eq*) = *eq_int*). (For the proof one uses the fact that all ground terms over SEQBIT of sort *int* can be reduced to a normal form $\langle L, \varepsilon\rangle$ or $\langle L, \varepsilon\&L\&a_1\&\ldots\&a_n\rangle$ or $\langle O, \varepsilon\&L\&a_1\&\ldots\&a_n\rangle$ with $a_i \in \{O, L\}$.)

Moreover, one can show that the parameterized specifications SET, LIST, and ARRAY_POINTER are parameter complete and parameter tolerant. Since by example 5.5 ARRAY_POINTER <~~~ SET proposition 6.2 and 6.3 can be applied and one obtains by horizontal composition: ARRAY_POINTER(SEQBIT) <~~~ SET(INT).

All implementation relations which can be constructed for the specifications in our example by horizontal and vertical composition are illustrated by the following diagram:



◊

7. Discussion

In the examples given above the formal parameter ELEM specifies not only the boolean values as observable but also all objects of sort *elem*. Hence it would be meaningless e.g. to construct the specification SET(SET(INT)) (with an appropriately defined equality operation for sets) since sets are not specified as observable and hence SET(INT) is not an admissible parameter of SET. As a solution there are (at least) two possibilities:
First, one could simply omit the axiom $Obs(x_{elem})$ in the formal parameter ELEM. Then, of course, all implementation relations from above remain valid. Compared with our example there is only one difference concerning the parameterized specifications LIST and ARRAY_POINTER since now the elements of lists can not more be observed via the operation *first* (but still are implicitly observable by the operations *eq* and *iselem*). This means that we have adopted a more abstract view which allows to abstract from the non observable properties of the elements. That way it would cause no problems to construct e.g. the following implementations: ARRAY_POINTER(ARRAY_POINTER(INT)) <~~~ LIST(ARRAY_POINTER(INT)) <~~~ LIST(LIST(INT)) <~~~ SET(LIST(INT)) <~~~ SET(SET(INT)).
The next possibility is to omit again the axiom $Obs(x_{elem})$ in the formal parameter ELEM but to introduce the axiom Obs(first(s)) in the body of the parameterized specifications LIST and ARRAY_POINTER (yielding parameterized specifications, say LIST' and ARRAY_POINTER'). Then still the implementation relations ARRAY_POINTER' <~~~ LIST' <~~~ SET hold but LIST' and ARRAY_POINTER' are not more parameter tolerant since they introduce more objects as observable than specified by the formal parameter. In fact, LIST' and ARRAY_POINTER' are not monotonic. (This is not surprising since an observational implementation SP1 of SP (where SP and SP1 are admissible parameters of LIST') may preserve only identities between terms of the observable sort *bool* whereas e.g. the implementation relation LIST'(SP1) <~~~ LIST'(SP) only holds if LIST'(SP1) preserves additionally all identities between terms of sort *elem*.) However, for the parameterized specifications SET and LIST' (resp. SET and ARRAY_POINTER') horizontal composition works since proposition 6.2 says that it is enough if one of the parameterized specifications involved is monotonic. Since SET is monotonic regardeless whether the axiom $Obs(x_{elem})$ belongs to the formal parameter or not one obtains for all admissible parameters SP and SP1 such that SP1 <~~~ SP, the implementation relation LIST'(SP1) <~~~ SET(SP) (resp. ARRAY_POINTER'(SP1) <~~~ SET(SP)).

A further issue to be discussed is more generally concerned with our notion of admissible parameter and its consequences with respect to the horizontal composition property. Consider e.g. a parameterized specification P, an admissible parameter SP of P, and an observational implementation SP1 of SP. If one wishes to implement P(SP) by P(SP1) this may be not feasible in our framework if SP1 is not an admissible parameter of P. To prevent this situation the notion of admissible parameter could be appropriately generalized such that observational implementations automatically preserve admissibility of parameters. From the observational point of view we suggest that an actual parameter should be considered as "observationally admissible" if it respects the observable behaviour specified by the formal parameter. Formally, this notion could be defined similarly to the observational implementation relation by requiring behaviour class inclusion wrt. the behaviour classes of the actual and the formal parameter. In contrast to observational implementations in this case the definition of behaviour classes has to be extended to non term generated algebras.

8. Concluding remarks

The present study shows that observational specifications provide a semantically well founded and flexible tool for the formal development of software. Based on the theory of observational specifications the observational implementation relation formalizes the intuitively clear idea that an implementation is correct if it produces correct observable output. It has been shown that observational implementations compose vertically and (under certain conditions) horizontally and hence are useful for the modular and stepwise construction of implementations. An advantage of the observational specification technique is their axiomatic description of behaviours which gives rise to the development of proof theoretic criteria for implementation relations and horizontal composability. Since it is highly desirable to support implementation proofs by machine a further step should be the development of algorithms for the verification of implementation relations. Based on the principle of context induction a first attempt into this direction has been undertaken in [Hennicker 88].

Acknowledgements
I would like to thank Martin Wirsing who supported this work by many ideas and valuable inspirations and Manfred Broy for many fruitful discussions. I gratefully acknowledge a number of useful comments made by the referees.

References

[Broy et al. 84]
M. Broy, C. Pair und M. Wirsing: A systematic study of models of abstract data types. *Theoretical Computer Science* **33**, 139-174 (1984).

[Broy et al. 86]
M. Broy, B. Möller, P. Pepper und M. Wirsing: Algebraic implementations preserve program correctness. *Science of Computer Programming* **7**, 1, 35-54 (1986).

[Burstall, Goguen 80]
R.M. Burstall, J.A. Goguen: The semantics of Clear, a specification language. *Proc. of Advanced Course on Abstract Software Specifications*, Kopenhagen. Springer Lecture Notes in Computer Science **86**, 292-332 (1980).

[Ehrig, Kreowski 82]
H. Ehrig, H.J. Kreowski: Parameter passing commutes with implementation of parameterized data types. In: M. Nielsen, E.M. Schmidt (eds.): *Proc.* ICALP 82, *9th Coll. on Automata, Languages and Programming*, Aarhus, July 1982. Springer Lecture Notes in Computer Science **140**, 197-211 (1982).

[Ehrig, Mahr 85]
H. Ehrig, B. Mahr: Fundamentals of algebraic specification 1. EATCS Monographs on Theor. Comp. Science, Vol. 6, Springer Verlag (1985).

[Ehrig et al. 82]
H. Ehrig, H.J. Kreowski, B. Mahr und P. Padawitz: Algebraic implementation of abstract data types. *Theoretical Computer Science* **20**, 209-263 (1982).

[Futatsugi et al. 85]
K. Futatsugi, J.A. Goguen, J.P. Jouannaud und J. Meseguer: Principles of OBJ2. *Proc. 12th* ACM *Symposium on Principles of Programming Languages*, New Orleans, 52-66 (1985).

[Ganzinger 83]
H. Ganzinger: Parameterized specifications: parameter passing and implementation with respect to observability. ACM Trans. on Prog. Lang. and Systems **5**, 3, 318-354 (1983).

[Geser, Hussmann 86]
A. Geser, H. Hussmann: Experiences with the RAP system – a specification interpreter combining term rewriting and resolution. In: B. Robinet, R. Wilhelm (eds.): *Proc.* ESOP 86, *Europ. Symp. on Programming*, Saarbrücken, March 1986. Springer Lecture Notes in Computer Science **213**, 339-350

[Giarratana et al. 76]
V. Giarratana, F. Gimona und U. Montanari: Observability concepts in abstract data type specification. In: A. Mazurkiewicz (ed.): *Proc.* MFCS 76, *5th Internat. Symp. on Mathematical Foundations of Comp. Science*, Gdansk, Sept. 1976. Springer Lecture Notes in Computer Science **45**, 576-587 (1976).

[Goguen, Burstall 80]
J.A. Goguen, R.M. Burstall: CAT, a system for the structured elaboration of correct programs from structured specifications. Technical report CSL-118, Computer Science Laboratory, SRI International

[Goguen, Meseguer 82]
J.A. Goguen, J. Meseguer: Universal realization, persistent interconnection and implementation of abstract modules. In: M. Nielsen, E.M. Schmidt (eds.): *Proc.* ICALP 82, *9th Coll. on Automata, Languages and Programming*, Aarhus, July 1982. Springer Lecture Notes in Computer Science **140**, 265-281 (1982).

[Hennicker 88]
R. Hennicker: Beobachtungsorientierte Spezifikationen. Dissertation, Fakultät für Mathematik und Informatik, Universität Passau (1988).

[Hennicker, Wirsing 85]
R. Hennicker, M. Wirsing: Observational specification: a Birkhoff-theorem. In: H.J. Kreowski (ed.): Recent Trends in Data Type Specification. *3rd Workshop on Theory and Appl. of Abstract Data Types*, Selected Papers. Informatik Fachberichte **116**, 119-135, Springer Verlag (1985).

[Nivela, Orejas 87]
Mª P. Nivela, F. Orejas: Initial behaviour semantics for algebraic specifications. *Proc. 5th Workshop on Algebraic Specifications of Abstract Data Types*, Gullane, September 1987, Springer Lecture Notes in Computer Science **332**, 184-207 (1988).

[Pepper 83]
P. Pepper: On the correctness of type transformations. Talk at *2nd Workshop on Theory and Appl. of Abstract Data Types*, Passau, May 1984.

[Reichel 81]
H. Reichel: Behavioural equivalence -- a unifying concept for initial and final specification methods. In: M. Arato, L. Varga (eds.): Math. Models in Comp. Systems, *Proc. 3rd Hungarian Computer Science Conf.*, Budapest, January 1981, 27-39 (1981).

[Sannella, Tarlecki 85]
D.T. Sannella, A. Tarlecki: On observational equivalence and algebraic specification. In: H. Ehrig, C. Floyd, M. Nivat, J. Thatcher (eds.): *Proc.* TAPSOFT 85, *Joint Conf. on Theory and Practice of Software Development*, Berlin, March 1985. Springer Lecture Notes in Computer Science **185**, 308-322 (1985).

[Sannella, Tarlecki 87]
D.T. Sannella, A. Tarlecki: Toward formal development of programs from algebraic specifications: implementations revisited. *Proc.* TAPSOFT 87, *Joint Conf. on Theory and Practice of Software Development*, Pisa, March 1986. Springer Lecture Notes in Computer Science **249**, 96-110 (1987).

[Sannella, Wirsing 82]
D.T. Sannella, M. Wirsing: Implementation of parameterized specifications. In: M. Nielsen, E.M. Schmidt (eds.): *Proc.* ICALP 82, *9th Coll. on Automata, Languages and Programming*, Aarhus, July 1982. Springer Lecture Notes in Computer Science **140**, 473-488 (1982).

[Schoett 87]
O. Schoett: Data abstraction and the correctness of modular programming. Ph.D. thesis, CST–42–87, Department of Comp. Science, University of Edinburgh (1987).

[Selman 72]
A. Selman: Completeness of calculii for axiomatically defined classes of algebras. *Algebra universalis* **2**,

[Wirsing 86]
M. Wirsing: Structured algebraic specifications: a kernel language. *Theoretical computer science* **42**, 123-249 (1986).