J. Symbolic Computation (1994) **18**, 41–81

Abstract Parametric Classes and Abstract Data Types defined by Classical and Constructive Logical Methods

PIERANGELO MIGLIOLI, UGO MOSCATO AND MARIO ORNAGHI

Department of Information Science, University of Milano (Italy)

(Received 1992)

We introduce a methodology to treat abstract data types (ADT), abstract parametric classes (APC) and subclasses, together with appropriate inheritance properties, by means of first order theories. The notion of a first order theory axiomatizing an ADT is based on the notion of isoinitial model and has been proposed by the authors in previous papers (Bertoni *et al.* (1979), Bertoni *et al.* (1983), Bertoni *et al.* (1984)). A theory formalizing an APC is seen, in this paper, as a theory T incompletely axiomatizing an ADT. Given a class C of ADT's, the class formalized by T can be seen (under suitable soundness conditions on T) as the class of the instances of T over C. An instantiation of T by an ADT I of C completes T into a T' formalizing an ADT I', which extends I and inherits the properties of the APC T.

We use both classical and constructive methods in the following sense: on the one hand, the semantics is based on classical model theory; on the other hand, the soundness of a consistent axiomatization can be analyzed by purely syntactical methods, in terms of provability within suitable constructive systems.

A theory T formalizing an APC (or an ADT) is not given by a list of axioms, but by a suitable "APC-expression", which explicitly or implicitly (but *effectively*) defines the axioms of T. We have APC-expressions to define APC's, to extend already defined APC's and to instantiate APC's (into ADT's or subclasses). We allow also "recurrence APC-expressions". At the end of the paper we give some examples showing how the proposed methodology works.

1. Introduction

In the late seventies, in the debate about the appropriate explanation of the intuitive notion of "abstract data type" (henceforth ADT), some difficulties have been put into evidence concerning the expressive power of ADT specifications and the possibility of getting, with such specifications, "recursive ADT's". To overcome these critical points, various more or less ad hoc solutions have been introduced by different approaches, among which the most popular one was the *initial algebra approach* of Goguen *et al.* (1976). In this frame, the *isoinitial models* as ADT's have been proposed (Bertoni *et al.* (1979), Bertoni *et al.* (1980), Bertoni *et al.* (1983)) in the attempt of providing a general solution simultaneously accounting for the expressive power of ADT specifications and for the recursiveness of ADT's. This proposal was connected with paradigms different from the ones involved in the previous approaches, namely:

0747–7171/94/010041 + 41 $08.00/0

a) the elimination of restrictions in the form of the axioms specifying an ADT, with the consequent choice of the full expressive power of first order languages and the loss of the "algebraic nature" of ADT's;
b) the definition of abstract data as "model-theoretic invariants" (Bertoni *et al.* (1979)), with an implicit proof-theoretic characterization of the decidability of the relations in an ADT.

The two points are related, since only a reasonable degree of expressiveness (including, in particular, the presence of negation) allows one to capture the notion of "abstract datum", together with the involved notion of decidability. However, the most prominent feature of the isoinitial model approach is just the replacement of the algebraic paradigm with the decidability paradigm. As a consequence, isoinitial models of axiomatizable theories turn out to be, so to speak, "recursive by definition", i.e., the isomorphism class of an isoinitial model always contains a (possibly many sorted) model whose finite carriers are finite sets of natural numbers, whose infinite carriers coincide with the set of the natural numbers, and whose functions and relations are general recursive (see, e.g., Bertoni *et al.* (1983)); this does not hold, in general, for the initial models of algebraic specifications.

On the other hand, the strength of the algebraic paradigm depends on general results of Universal Algebra, according to which an initial model exists for every algebraic specification, possibly extended to include Horn clauses; this is not true for the isoinitial model approach, which lacks an *effective* necessary and sufficient condition in order to distinguish the first order theories with isoinitial models from the other ones. Thus, partly for the lack of such a general condition, partly for a greater familiarity with algebraic tools than with model-theoretic or proof-theoretic tools, isoinitial models have not been taken into the appropriate consideration, despite their acknowledged nice properties. This situation can be summarized by Wirsing's opinion that (besides the three main approaches to ADT's) "a fourth approach which is mainly of theoretical interest uses isoinitial algebras" (see Wirsing (1990)).

Yet, effective sufficient conditions assuring the existence of isoinitial models can be found for wide classes of first order theories. Moreover, these theories may be very powerful and contain such important principles as induction; e.g., a strong theory such as first order Peano Arithmetic has such a paradigmatic isoinitial model as the standard model of the natural numbers. Also, every recursive structure can be seen as the isoinitial model of a simple theory, namely, a recursive diagram of the structure. Finally, in the full first order frame initiality too lacks an effective necessary and sufficient condition of existence, since, in general, an arbitrary first order theory has no initial models.

On the other hand, even if recursiveness is not the intrinsic property of initiality, various techniques have been developed in order to get recursive ADT's from algebraic specifications. Thus, the advantages and the drawbacks of the isoinitial and of the algebraic specifications are, to some extent, balanced. In this sense, a more accurate comparison of the two attitudes on the ground of possible practical applications should clarify further important aspects. However, the development of the isoinitial model approach, after the publication of the first papers, has run in a different direction, outside the traditional field of ADT's.

In the subsequent investigation the authors have explored the possibility of capturing by means of *constructive* formal systems the main features of isoinitiality. This has been motivated, on the one hand, by the circumstance that the analysis of isoinitiality carried out in *classical* Model Theory in terms of the notion of "model-theoretic invariant" (with

the related quotient construction introduced in Bertoni *et. al* (1979) to characterize all the isoinitial models) seemed to be related to typical ideas of constructive semantics. On the other hand, as it is well-known, constructive provability of suitable formulas describing functions (relations) implies the computability of the involved functions (the decidability of the involved relations); thus, it seems to be strongly connected with the paradigmatic properties of isoinitial models.

The results of this investigation, indeed, have confirmed these insights. In Bertoni *et al.* (1984), Miglioli *et al.* (1988) and Miglioli *et al.* (1989), not only (reachable) isoinitiality has been presented in terms of constructive provability, but also a kind of extension result for ADT's has been put into evidence which has no counterpart in classical logic. To give a simple example, let a formula such as $\forall x \exists! y A(x, y)$ be *classically* provable from a theory T with a reachable isoinitial (initial) model I. Then the formula $A(x, y)$ implicitly defines, for every model of T, a function associating, with every α of the appropriate carrier, a unique β such that $A(\alpha, \beta)$ holds; in particular, $A(x, y)$ defines a function f_I on the isoinitial (on the initial) model I. But, parallel to the fact that a classical proof of $\forall x \exists! y A(x, y)$ cannot guarantee the recursiveness of the function f_I, we cannot be sure that the definitory extension $T \cup \{\forall x A(x, f(x))\}$ (with f a new function symbol) has an isoinitial model (even an initial model!), in particular that the expansion of I with f interpreted as the function f_I is an isoinitial (an initial) model of $T \cup \{\forall x A(x, f(x))\}$. Only if $\forall x \exists! y A(x, y)$ can be proved in a *constructive* formal system whose deductive apparatus is weaker than the classical one and whose mathematical part coincides with T, we can be sure that $T \cup \{\forall x A(x, f(x))\}$ has an isoinitial (an initial in the other case!) model which is an expansion of I where the symbol f is interpreted by f_I (notice that in this case, where the considered provability relation is constructive, the notions of isoinitiality and of initiality are still defined in terms of the *classical* models of T).

While the above example involves a definitory (hence a conservative) extension of a theory, more powerful results have been found allowing one to get, in a constructive setting, expansions of isoinitial models related to possibly non-conservative extensions of the corresponding theories. Also, a great effort has been made in order to single out superintuitionistic logical principles to build up more and more powerful constructive formal systems whose mathematical parts are theories with reachable isoinitial models; as a matter of fact, the stronger the constructive formal system in hand, the stronger become, at least in principle, the expansion results which can be obtained using constructive methods. Thus, an analysis of the relative constructive compatibility of various mathematical and logical principles (some of which were not known in the literature) has been carried out giving rise to various classes of "large constructive systems" whose mathematical parts are theories with reachable isoinitial models (see Miglioli *et al.* (1988) and Miglioli *et al.* (1989)). These researches should be of potential interest also for the traditional field of ADT's, even if their practical applications have not been fully explored and the results have been presented mainly on a theoretical ground.

Near the above developments, involving constructiveness merely as a tool to analyze ADT's, other aspects have been taken into account, in connection with the researches in constructive program synthesis, where the authors were engaged. In this frame, various techniques to extract the "algorithmic content" of proofs belonging to suitable constructive systems have been proposed, and the paradigm "proofs as programs" has progressively emerged. We quote, e.g., papers such as Degli Antoni *et al.* (1974), Degli Antoni *et al.* (1975), Sato (1979), Goto (1979), Goad (1980), Miglioli and Ornaghi (1981), Martin-Löf (1982), Bates and Constable (1985), Constable *et al.* (1986), Hayashi and Nakano

(1988), Henson (1989) and Bundy *et al.* (1990), some of which having a more theoretical character and involving a logical background of Constructive Semantics and Proof Theory, some others being concerned with implementations. Reports on a partial implementation by the authors of a system named PAP (Proofs As Programs) have been given in Bresciani *et al.* (1986), and in Miglioli *et al.* (1988 bis); a thorough discussion on some critical points and perspectives connected with the implementation of PAP is given in Miglioli *et al.* (1992).

Thus, merging their work on program synthesis with the one on ADT specification, an investigation has been undertaken by the authors concerning the possibility of treating in a unified constructive context *abstract data types* and *abstract programs on an ADT*, the latter being proofs which can be built up within suitable formal systems whose mathematical parts are theories axiomatizing ADT's (reachable isoinitial models). In other words, the authors have come to the proposal of *setting up a theory of abstract data types for constructive program synthesis.*

Of course, in this area the problem of the axiomatization of specific data structures is not disregarded (induction principles are taken into account, e.g., in Goad (1980), Bates and Constable (1985) and Bundy *et al.* (1990), while Goad (1980) proposes the use of Harrop axioms). However, the question has not been raised in systematic terms, perhaps because of the influence of intuitionistic tradition in constructivism. In this sense, matters of axiomatization are much less flexible according to intuitionistic doctrine than to classical one; and more than that, a "formal theory of axiomatization" (independent to a good extent of the nature of the axiomatized objects) is not even conceivable by the orthodox intuitionistic school, which has left only a list of paradigmatic examples (the theory of natural numbers, the theory of species, the theory of real numbers, an so on). On the contrary, we believe that in massive practices such as the ones involved in Computer Science, a "formal theory of axiomatization" is needed. To this aim, we propose a *mixed use of classical* (model-theoretic) *and constructive* (proof-theoretic and semantic) *methods* to tackle in a unified way ADT specification and program synthesis. Indeed, we believe that even in program synthesis the involved constructive algorithmic aspects must not obscure the possibility of getting a simple classical reading of formulas (specifications). We aim to show that our mixed approach based on isoinitiality (in the context of first order axiomatizations powerful enough) can be very fruitful. In the field of program synthesis, other authors use constructive methods quite compatible with classical semantics (see, e.g., Hayashi and Nakano (1988) and Voronkov (1990)). However, no author in this frame combines classical and constructive techniques in such an integrated way as we do.

Our most recent work on ADT's in program synthesis is connected with the development of the above quoted system PAP, which we are redesigning. PAP has been initially designed to experiment with the paradigm "proofs as programs" for particular first order systems such as Intuitionistic Arithmetic. Successively, we have taken into account the class of systems with reachable isoinitial models, with mathematical axioms given by Harrop formulas and instances of induction principles and with the logical part consisting of intuitionistic logic enriched by Kuroda's Principle $\forall x \neg\neg A(x) \rightarrow \neg\neg \forall x A(x)$ (see Subsection 2.4). In this sense, only *complete* axiomatizations were admitted: on the one hand a mathematical axiomatization had to specify in *full detail* a (recursive) isoinitial ADT and to *implicitly* provide the primitive computational resources (the algorithms to decide the relations of the ADT and to compute its functions) to be used in order to make more complex algorithms (the constructive proofs definable in the formal system); on the other hand, a special *proof-language* had to be used to make proofs in the con-

structive system and to attach to them a double deductive and algorithmic meaning. A proof-checker had to automatically analyze the correctness of a proof (which had not, in turn, to be built up automatically), where the proof-language had been designed in order to make as explicit as possible its "algorithmic content" (its interpretation as a program), still maintaining a clear reading of its deductive task.

In line with the above discussion, the ultimate semantics involved in formulas and in proofs was classical. In this frame, a provable formula was seen as a formula whose (classical) truth can be *algorithmically* (or *effectively*) justified, while a proof of it was seen as an algorithm to carry out such a justification. For, the rules of the proof-language were equipped with an *operational semantics* allowing to correctly perform such effective evaluations of truth (see, e.g., Miglioli *et al.* (1988 bis) and Miglioli *et al.* (1992)).

At the same time, the computational mechanism involved in the effective analysis of truth allowed to read formulas as program specifications and their proofs as programs performing these specifications. For instance, a proof of a formula $\forall x \exists y A(x, y)$ can also be seen as a proof of the open formula $\exists y A(x, y)$, which can be "executed" (as a program) to get, for every closed term t, an effective evaluation of the truth of $\exists y A(t, y)$; being effective, such an evaluation *justifies* the truth of $\exists y A(t, y)$ in terms of the truth of a suitable formula $A(t, t')$, providing also a term t' such that $A(t, t')$ holds; thus, a function can be computed associating, with every element α of an ADT I (represented by t) an element β of I (represented by t') such that $A(\alpha, \beta)$ holds in I.

It is worth mentioning, with this example, that the notion of program and the use of ADT specifications involved in PAP are quite different from the ones involved in Logic Programming. For, in PAP programs are proofs of general facts (involving infinitely many instances), while the proof-language (i.e., the deductive system) can be seen as the programming language, a part of which is given by the axioms of the ADT specification. On the other hand, in Logic Programming (and in the Algebraic Approach with rewriting systems) axioms work both as the definition of a general problem and as the program to solve it, while proofs work as execution sequences.

Our former experience with PAP has shown that, to use such a system in a realistic way, the greatest difficulties are not connected with the interpretation of proofs as programs, but in the effort needed to build up proofs and in the massive production of theories axiomatizing *complete* ADT's. To overcome the second critical point, we have found more appropriate the introduction of a specification mechanism allowing to group into classes ADT theories, i.e., to make *parametric* specifications, giving rise to *abstract parametric classes* (henceforth, APC's).

Our notion of APC is taken from works in the area of abstract data types and object oriented programming such as Goguen (1984), Ehrig and Mahr (1985), Goguen and Meseguer (1987), Ehrig and Mahr (1989), Goguen and Burstall (1992). These works have also inspired the basic concepts of our treatment such as *parameter passing morphism* and *instantiation*. However, our investigation has been carried out in strict connection with the research on PAP and our constructive attitude. This has given rise to a formalization based on notions which can be expressed only in constructive terms, even if the results obtained seem to be relevant also from a classical point of view.

According to our characterization, an APC can be seen from a double point of view: from the classical point of view it turns out to be an *incomplete* axiomatization of ADT's to be *instantiated* by means of theories with (isoinitial) ADT's, giving rise to a class of *complete* axiomatizations of ADT's; from the constructive point of view, it gives rise (under appropriate constructive inference rules) to a *virtually constructive* formal system

to be made *actually constructive* by means of instantiations with theories with (isoinitial) ADT's. As an axiomatization becomes complete after instantiations, the corresponding formal system with constructive inference rules becomes constructive and conversely. Such a parallelism between complete axiomatizations and actual constructiveness is the main feature of our specification mechanism. It gives rise to effective syntactical characterizations of APC's *correctly behaving with respect to instantiations*. It also provides *effective extensions* allowing to expand APC's into richer APC's, in such a way that the instances of the parent class inherit the new functions and relations provided by the class expansion, according to the well-known commutative diagram. The effectiveness of these results has been codified by special *APC-expressions* to implicitly, but *automatically*, define APC's, instantiations of APC's and expansions of APC's. The APC-expressions are intended as a step to characterize the ADT-specification methodology involved in PAP.

Now, the paper has three main purposes:

1) to present a part of the actual project PAP, i.e., to explain its ADT-specification mechanism;
2) to show that our results are interesting for the ADT area even if constructiveness is not taken as the main concern;
3) to provide various examples of our method and to show that it can be interesting also from the point of view of applications.

The paper is so organized. Section 2 contains the preliminary notions and, after the introductory Subsection 2.1, provides: a review of the main feature of isoinitiality and a comparison with initiality, in the frame of classical logic (Subsection 2.2); a characterization of reachable isoinitial models, a definition of ADT as a reachable isoinitial model and examples of ADT's (Subsection 2.3); an illustration of constructive formal systems (Subsection 2.4); some insights on the use of constructive systems and of induction principles in our methodology (Subsection 2.5). Section 3 introduces notions such as parameter passing, instantiations, APC expansions and related expansion inheritance, and provides a first discussion on the use of constructiveness to guarantee the soundness of the introduced operations; also APC expressions are introduced to effectively describe such operations. Section 4 provides the results on which our methodology is based. Section 5 contains various examples of APC's defined according to our methodology; also APC's defined by recurrence are included. Finally, Section 6 gives some results on APC-specifications by recurrence.

2. Preliminary notions

In this section we first introduce some basic notions and notations. Then, in Subsection 2.2, we recall the main theoretical results concerning our previous treatment of isoinitial models in the frame of classical logic (Bertoni *et al.* (1979), Bertoni *et al.* (1980), Bertoni *et al.* (1983)). This provides an introduction to our notion of ADT as a reachable isoinitial model, given in Subsection 2.3; in this subsection we also present some examples oriented toward the applications of ADT's considered in this paper. Since, as we will see in Section 4.1, the main features of the isoinitial approach can be captured in a constructive setting, Subsection 2.4 will be devoted to the explanation of some basic notions involving constructive first order systems. The final subsection will give some

insights about the use of constructive systems in our methodology, and will discuss the role of induction in this frame.

2.1. BACKGROUND AND NOTATIONS

A many sorted signature will be represented by a quadruple $\Sigma = \langle S, C, F, R\rangle$ such that:

- S is a set of *sort symbols*, we will indicate by $s, s_1, s_n, \ldots$;
- C is a set of *constant declarations*, of the kind $c : s$, where c is a constant symbol and s is a sort symbol (the *sort of* c);
- F is a set of *function declarations*, of the kind $f : \underline{a} \rightarrow s$, where f is a *function symbol*, $\underline{a} = s_1 s_2 \cdots s_n$ is a string of sort symbols (the *arity of* f) and s is a sort symbol (the *sort of* f);
- R is a set of *relation declarations*, of the kind $r : \underline{a}$, where r is a *relation symbol* and $\underline{a} = s_1 s_2 \cdots s_m$ is a string of sort symbols (the *arity of* r).

We will use overloaded symbols (in particular, overloaded identity $=$), where a function or a relation symbol is *overloaded* if it appears in more than one declaration. Overloaded constants will be *forbidden*; also, function declarations $f : \underline{a} \rightarrow s$ and $f : \underline{a} \rightarrow s'$, with identical $f, \underline{a}$ and different s, s', will be *forbidden*.

$Term_\Sigma$ will be the set of terms built up in the usual way, starting from Σ (and a prefixed denumerable set X of sorted variables). If no forbidden declaration occurs, every term has a unique sort. We call L_Σ the first order language generated by the signature Σ (and the set X chosen for variables).

Σ-structures are defined in the usual way, provided that an overloaded function or relation symbol is interpreted into many functions or relations, one for each declaration where it appears. A Σ-structure will be indicated by a couple:

$$A_\Sigma = (\mathbf{A}, \mathbf{i})$$

where $\mathbf{A} = \{\mathbf{A}_s | s \in S\}$ is an S-indexed family of *non-empty* sets, called the *carriers* of the sorts of S, and $\mathbf{i}$ is the interpretation of C, F, R. In A_Σ terms and formulas are interpreted in the usual way. $Struct(\Sigma)$ will indicate the class of the Σ-structures.

We say that A_Σ is reachable iff, for every sort s, all the elements of its carrier are denoted by closed terms. By

$$A_\Sigma \models F(\underline{\alpha})$$

we mean that the formula $F(\underline{x})$ is true in A_Σ under the assignment $\underline{x} = \underline{\alpha}$ (for F closed, $\underline{\alpha}$ is omitted).

Let $A_\Sigma = (\mathbf{A}, \mathbf{i})$, $B_\Sigma = (\mathbf{B}, \mathbf{k})$ be Σ-structures; *homomorphisms* and *isomorphisms* $h : A_\Sigma \rightarrow B_\Sigma$ are defined in the usual way (h is a S-indexed family of functions $h_s : \mathbf{A}_s \rightarrow \mathbf{B}_s$).

An *isomorphic embedding* is an injective homomorphism such that, for every relation declaration $r : s_1 \cdots s_m$, $\langle \alpha_1, \ldots, \alpha_m\rangle \in \mathbf{i}(r)$ iff $\langle h_{s_1}(\alpha_1), \ldots, h_{s_m}(\alpha_m)\rangle \in \mathbf{k}(r)$.

First order Σ-theories (i.e., with language L_Σ) and their models are defined in the usual way. A Σ-theory T_Σ is said to be *axiomatizable* iff T_Σ is a recursively enumerable set of (closed) formulas. As it is well-known in the literature, for every axiomatizable Σ-theory T_Σ, there is a Σ-theory T'_Σ which is a *recursive presentation* of T_Σ in the following

sense: a) T'_Σ is a recursive set of (closed) formulas; b) the set of formulas classically provable from T_Σ coincides with the set of formulas classically provable from T'_Σ. We are interested in effective operations on theories and, in this paper, we always work with recursive presentations. In this frame, the form of the axioms is relevant. For example, it is well-known that *equational* and *Horn presentations* always admit initial models. In our approach we will consider a wider class of presentations, where the existence of an isoinitial model can be studied by effective methods, using constructive proofs. In all the presentations (henceforth called theories) considered in this paper, we will work in *first order logic with identity*, where overloaded identity $=$ and the related identity axioms are always included.

Finally, to work with different signatures and theories we will use the following notions, well-known in the literature (Goguen *et al.*, (1992)).

A *signature morphism* $\mu : \Sigma \to \Sigma'$ maps the symbols of Σ into symbols of Σ', preserving arities and sorts. The morphism μ induces the forgetful functor $\rho : \mathit{Struct}(\Sigma') \to \mathit{Struct}(\Sigma)$ such that, for every Σ'-structure $A'_{\Sigma'} = (\mathbf{A}', \mathbf{i}')$, $\rho(A'_{\Sigma'})$ is the Σ-structure $A_\Sigma = (\mathbf{A}, \mathbf{i})$ so defined:

- for every sort s, $\mathbf{A}_s = \mathbf{A}'_{\mu(s)}$;
- for every constant declaration $c : s$, $\mathbf{i}(c : s) = \mathbf{i}'(\mu(c) : \mu(s))$;
- for every function declaration $f : \underline{a} \to s$, $\mathbf{i}(f : \underline{a} \to s) = \mathbf{i}'(\mu(f) : \mu(\underline{a}) \to \mu(s))$;
- for every relation declaration $r : \underline{a}$, $\mathbf{i}(r : \underline{a}) = \mathbf{i}'(\mu(r) : \mu(\underline{a}))$.

We call A_Σ the *reduct* of $A'_{\Sigma'}$ and $A'_{\Sigma'}$ an *expansion* of A_Σ. If μ is injective, we call it an *expansion morphism*; for expansion morphisms, the notions of reduct and expansion coincide with the usual ones (see, e.g., Chang and Keisler (1973)). Intuitively, an expansion $A'_{\Sigma'}$ of A_Σ contains the new interpretations of the new symbols (the ones not in the range of μ), while the old symbols (in the range of μ) are interpreted as before (as in A_Σ).

A theory morphism $\mu : T_\Sigma \to T'_{\Sigma'}$ is a signature morphism $\mu : \Sigma \to \Sigma'$ such that, for every axiom A of T_Σ, its translation $\mu(A)$ is provable in $T'_{\Sigma'}$ using *classical logic*.

2.2. FIRST ORDER THEORIES FORMALIZING ISOINITIAL MODELS

Our formalization of ADT's is based on the following notion of isoinitial model of a first order Σ-theory T_Σ:

> I_Σ is an *isoinitial model* of T_Σ iff, for every model A_Σ of T_Σ, there is a *unique isomorphic embedding* $h : I_\Sigma \to A_\Sigma$.

For our discussion it is useful to introduce also the following notion of initiality in the *full* first order context:

> J_Σ is an *initial model* of T_Σ iff, for every model A_Σ of T_Σ, there is a *unique homomorphism* $h : J_\Sigma \to A_\Sigma$.

It is worth mentioning that, in the full first order frame, there are theories without initial and isoinitial models.

Comparing the two notions, we have that both capture "abstractness", in the paradigmatic sense of the literature on ADT's. In other words, we have:

an initial model J_Σ of a first order theory T_Σ (if it exists) is unique up to isomorphisms;
an isoinitial model I_Σ of a first order theory T_Σ (if it exists) is unique up to isomorphisms.

We also have:

if J_Σ and I_Σ are respectively an initial and an isoinitial model of the same theory, then J_Σ and I_Σ are isomorphic, i.e., J_Σ is also isoinitial and I_Σ is also initial.

On the other hand, initiality and isoinitiality are independent notions in the following sense:

there are theories with initial models and without isoinitial models, and there are theories with isoinitial models and without initial models.

The notion of ADT, as given, e.g., in Wirsing (1990), requires the reachability of the involved structure. From this point of view, in the full first order frame initial and isoinitial models can be seen as *virtually reachable* models, as a consequence of the following fact:

PROPOSITION 2.1. *(I) Let J_Σ be an initial model of T_Σ. Then, for every carrier $\mathbf{J}_s$ and every $\alpha \in \mathbf{J}_s$, there is a positive quantifier free formula (i.e. a formula without quantifiers, negation and implication) $\Delta^+_\alpha(x, \underline{y})$ of L_Σ (x a variable, $\underline{y}$ a possibly empty set of variables) such that the following conditions hold:*

1) $\exists! x \exists \underline{y} \Delta^+_\alpha(x, \underline{y})$ is classically provable from T_Σ;
2) for every model A_Σ of T_Σ, if h is the unique homomorphism from J_Σ to A_Σ, then $A_\Sigma \models \exists \underline{y} \Delta^+_\alpha(h(\alpha), \underline{y})$.

(II) Let I_Σ be an isoinitial model of T_Σ. Then, for every carrier $\mathbf{I_S}$ and every $\alpha \in \mathbf{I_S}$, there is a quantifier free formula (possibly with negations and implications) $\Delta_\alpha(x, \underline{y})$ of L_Σ such that the following conditions hold:

1') $\exists! x \exists \underline{y} \Delta_\alpha(x, \underline{y})$ is classically provable from T_Σ;
2') for every model B_Σ of T_Σ, if h is the unique isomorphic embedding from I_Σ to B_Σ, then $B_\Sigma \models \exists \underline{y} \Delta_\alpha(h(\alpha), \underline{y})$.

The formulas Δ^+ and Δ, satisfying respectively Point (I) and Point (II) of Proposition 2.1, are called *positive abstract datum* and *abstract datum* respectively (see Bertoni *et al.* (1979); see also Bertoni *et al.* (1983) for positive abstract data). Given an initial model J_Σ, one can pass from virtual to actual reachability as follows: for every positive abstract datum $\Delta^+_\alpha(x, \underline{y})$, one enriches Σ by a new constant c_α, in such a way that c_α is different from $c'_{\alpha'}$ if the corresponding positive abstract data are; moreover, for every c_α inserted in Σ, one enriches T_Σ by the (inessential) definitory axiom $\exists \underline{y} \Delta^+_\alpha(c_\alpha, \underline{y})$. In a similar way, using abstract data, one gets the reachability of the isoinitial models. But here one can get more.

As a matter of fact, let I_Σ be an isoinitial model of T_Σ, let $\mathbf{I}_s$ be a carrier of I_Σ, let $\alpha, \beta \in \mathbf{I}_s$, and let $\Delta_\alpha(x, \underline{y})$ and $\Delta_\beta(x', \underline{y'})$ be associated abstract data; then the following fact holds:

PROPOSITION 2.2. *If* $\alpha \neq \beta$, *then* $\exists x \exists x' \exists \underline{y} \exists \underline{y}' (\neg x = x' \wedge \Delta_\alpha(x, \underline{y}) \wedge \Delta_\beta(x', \underline{y}'))$ *is classically provable from* T_Σ.

According to Propositions 2.1 and 2.2, if α and β are different elements of a carrier $\mathbf{I}_s$ of an isoinitial model I_Σ of T_Σ, then $h(\alpha)$ is different from $h(\beta)$ in every model A_Σ of T_Σ, $h(\alpha)$ and $h(\beta)$ being as in Point II) of Proposition 2.1. This is the sense according to which the elements of an isoinitial model are *model-theoretic invariants* according to Bertoni *et al.* (1979), a property which is not satisfied, in general, by the elements of an initial model.

Thus, one can (inessentially) extend a theory T_Σ with an isoinitial model I_Σ into a theory $T'_{\Sigma'}$ with an isoinitial model $I'_{\Sigma'}$, so as to satisfy the following properties:

a) if T_Σ is axiomatizable, then $T'_{\Sigma'}$ is;
b) $\Sigma' - \Sigma$ is a set of new constants denoting all the elements of $I'_{\Sigma'}$;
c) if c and c' are different constants of $\Sigma' - \Sigma$, then the corresponding elements of $I'_{\Sigma'}$ are different;
d) every model $A'_{\Sigma'}$ of $T'_{\Sigma'}$ is the expansion of some model A_Σ of T_Σ;
e) every model A_Σ of T_Σ is the reduct of some model $A'_{\Sigma'}$ of $T'_{\Sigma'}$.

Here the crucial point is c), which allows to get reachable isoinitial models where *every element is denoted by a term in normal form* (the constant of $\Sigma' - \Sigma$ corresponding to it). The possibility of obtaining Point c) depends on the possibility of proof- theoretically distinguishing the different elements of an isoinitial model, i.e., on Proposition 2.2. As a matter of fact, we can group abstract data into equivalence classes, where two abstract data of the same equivalence class provably characterize the same element of an isoinitial model; then, by Proposition 2.2, we can proof-theoretically check whether different abstract data belong to different classes, so as to prevent the introduction of couples of definitions of the form $\exists \underline{y} \Delta(c, \underline{y})$ and $\exists \underline{y}' \Delta'(c', \underline{y})$, with c, c' different constants and Δ and Δ' different formulas belonging to the same equivalence class. This cannot be made, in general, in the case of initial models, where an extension $T'_{\Sigma'}$ of T_Σ satisfying the above Points b), c) and d) (taking "initial" in place of "isoinitial") may fail to satisfy Point e) and, in addition, Point a). In other words, *to get reachable initial models with terms in normal form, one may affect the class of models in an essential way and may loose, moreover, the axiomatizability of the theory* (this happens when the initial model is not recursive, a possibility which is prevented in the case of isoinitial models).

This concludes our review of the main theoretical features of initial and isoinitial models in the full first order frame (for a more comprehensive discussion, see Bertoni *et al.* (1983)). According to the above, only isoinitial models are *virtually reachable in the strong sense*, i.e., the corresponding theories can be extended into theories with reachable isoinitial models whose elements are denoted by terms in normal form. This feature is strictly connected with recursiveness, which is the paradigmatic property of isoinitiality.

2.3. ABSTRACT DATA TYPES AS REACHABLE ISOINITIAL MODELS

According to the results explained in the previous subsection, the characterization of an ADT as an isoinitial reachable model seems to be well justified on the theoretical ground. In this line, we give the following definition.

A Σ-theory T_Σ *completely formalizes* an ADT I_Σ iff I_Σ is a *reachable isoinitial* model of T_Σ.

We point out that, when reachable models are considered, initiality and isoinitiality are no longer independent notions. For, one easily proves the following:

PROPOSITION 2.3. *If T_Σ completely formalizes an ADT I_Σ, then I_Σ is initial too.*

In narrow contexts such as the algebraic ones, reachable initial models always exist, while no necessary and sufficient condition can be found for isoinitiality. In contrast, in the full first order frame Theorem 2.1 below gives a *necessary and sufficient condition for reachable isoinitiality*, which can be used in an *effective* way in a wide class of cases and has no counterpart for initiality.

We say that T is *atomically complete* iff

$$T + CL \vdash A \text{ or } T + CL \vdash \neg A,$$

for every closed atomic formula A (here "$T + CL \vdash \ldots$" means provability from T using classical logic). Atomic completeness is needed to state our necessary and sufficient condition, whose proof is implicitly contained in Bertoni *et al.* (1984):

THEOREM 2.1. *T completely formalizes an ADT iff T has a reachable model and T is atomically complete.*

Theorem 2.1 provides another way of building up a *recursive isoinitial model* of T as follows:

- every element of each carrier is a recursive equivalence class $[t]$ of ground terms and every class $[t]$ contains a unique normal term t;

- every relation r is *decided* by T, i.e., $r([t_1], .., [t_n])$ holds if $r(t_1, .., t_n)$ is classically provable from T and $r([t_1], .., [t_n])$ does not hold if $\neg r(t_1, .., t_n)$ is classically provable from T.

Now we list some interesting examples of theories which turn out to completely formalize isoinitial ADT's as an almost immediate consequence of Theorem 2.1.

First Order Peano Arithmetic PA. PA is the well-known first order axiomatization of Peano Arithmetic, with one sort N and signature containing the declarations

$$0 : N, \quad s : N \to N, \quad + : N \times N \to N, \quad * : N \times N \to N, \quad =: N \times N.$$

Indeed, the standard structure of the natural numbers is a reachable model and, by the atomic completeness of PA, it is an isoinitial model of PA.

Constructors theories. Let C be a set of constant and function symbols of some signature, we call "constructors". If, for every sort s, the set of the ground terms of sort s is not empty, then the term-algebra generated by C is an isoinitial model of the theory:

$$Constructors(C) = ID \cup Inj(C)$$

where ID are the *identity axioms* (always included in our theories) and $Inj(C)$ is the set containing the following *injectivity axioms*:

a) $\neg c = c'$ (for any two different constants c and c' of C), $\forall \underline{x} : \neg c = f(\underline{x})$ (for any constant c and any function symbol f of C), $\forall \underline{x}, \underline{y} : \neg f(\underline{x}) = g(\underline{y})$ (for any two function symbols f and g of C);
b) $\forall x_1, \ldots, x_n, y_1, \ldots, y_n : (f(x_1, .., x_n) = f(y_1, .., y_n) \rightarrow x_1 = y_1 \wedge \ldots \wedge x_n = y_n)$ (for every function symbol f of C).

Indeed, *Constructors*(C) is atomically complete and the term algebra is a reachable model of it. *Constructors*(C) is known in the literature of Logic Programming as the equality theory of C (see, e.g., Lloyd (1987)). Also the completion of logic programs gives rise to axiomatizations of isoinitial ADT's, as follows.

Completion theories. Let P be a definite logic program and *Comp*(P) be its completion (see Lloyd (1987)). If P halts for every ground goal, then *Comp*(P) completely formalizes as an ADT the minimal Herbrandt model of P.

The isoinitiality of the minimal Herbrandt model follows from its reachability and from the fact that halting for every ground goal implies atomic completeness. This result can be extended to the completion of normal programs for which negation as failure works in a correct and complete way for ground goals.

Completed equational theories. Let E be an equational theory with a signature Σ such that the set of the ground terms of sort s is not empty for every s. By its completed theory *Comp*(E) we mean the following set of axioms:

$$Comp(E) = E \cup \{\neg t = t' | t, t' \text{ are ground and } E \not\vdash t = t'\}$$

An initial model of E is an isoinitial model of *Comp*(E). Notice that *Comp*(E) is a recursive axiomatization iff equality of ground terms is decidable.

More generally, whenever a recursively presented first order theory with a decidable set of provable atomic formulas and with a reachable model is given, one can complete it into a recursively presented theory completely formalizing an ADT. This fact shows the generality of the method and its strict connection with recursiveness.

The above examples of first order theories axiomatizing isoinitial ADT's can be enriched by various kinds of induction principles, as we will see in Subsection 2.4. Induction is a key point in our methodology, since it allows to use constructive methods in a relevant and powerful way.

2.4. CONSTRUCTIVE AND SEMICONSTRUCTIVE FIRST ORDER SYSTEMS

In this paper we will consider *constructive* and *classically sound* formal systems $S = T + L$, where:

- the notation "$T + L$" means that the system S consists of a mathematical part T (a first order theory which, interpreted according to classical semantics, has an isoinitial model) and an *intermediate pseudo logic* L, i.e. a set L of classically valid formulas closed under modus ponens and generalization –but not necessarily

under substitution- such that $INT \subseteq L \subseteq CL$, INT and CL being intuitionistic and classical logic respectively;
- the sense according to which S is *constructive* is that S satisfies the following disjunction property DP and explicit definability property EDP:

(DP) $S \vdash A \vee B$ and $A \vee B$ is closed $\Rightarrow$ $S \vdash A$ or $S \vdash B$
(EDP) $S \vdash \exists x C(x)$ and $\exists x C(x)$ is closed $\Rightarrow$ $S \vdash C(t)$ for some closed t

- the sense according to which *classical soundness* is assumed is that $T + L$ is consistent iff $T + CL$ is.

Classical soundness is automatically guaranteed by using intermediate pseudo logics L containing the (pseudo) logic IKA so characterized: IKA is obtained by adding to INT (intuitionistic predicate logic with identity) the following axioms:

(K) $\forall x \neg\neg A(x) \rightarrow \neg\neg \forall x A(x)$ (Kuroda principle, see Gabbay (1981), Troelstra (1973))
(A) $\neg\neg A \rightarrow A$ for any *atomic* A.

Of course, the restriction to the atomic formulas in axiom (A) prevents the validity (in IKA) of the non-constructive principle

$$\neg\neg H \rightarrow H$$

for any formula H.

We remark that the addition of (K) alone to INT is sufficient to obtain classically sound systems.

Also, classically sound systems corresponding to the following weaker notions of constructiveness could be used.

A system $S = T + L$ is *semiconstructive* iff it satisfies the following *weak disjunction property* WDP and *weak explicit definability property* WEDP:

(WDP) $S \vdash A \vee B$ and $A \vee B$ is closed $\Rightarrow$ $T + CL \vdash A$ or $T + CL \vdash B$

(WEDP) $S \vdash \exists x C(x)$ and $\exists x C(x)$ is closed $\Rightarrow$ $T + CL \vdash C(t)$ for some closed t.

A system $S = T + L$ contained in some (fully) constructive system $S' = T + L'$, with $L \subseteq L'$ and the *same* mathematical part T, is said to be T-subconstructive: it turns out that any *subconstructive* system is semiconstructive (but the converse does not hold).

We look at subconstructiveness and, more generally, at semiconstructiveness as one of the two extremes within which a constructive point of view may range, the other being strong constructiveness, where (roughly speaking): by a *strongly constructive system* we will mean any system S such that any proof of $A \vee B$ in S ($A \vee B$ closed) contains sufficient information to build up a proof of A in S or a proof of B in S, and the like for $\exists x A(x)$.

Strong constructiveness is appropriately treated in a proof-theoretical attitude, while subconstructiveness and semiconstructiveness generally involve simpler model-theoretic aspects. From the point of view of program specification and construction, the former is needed in contexts such as program synthesis, where proofs are taken as programs and must give rise to effective computations; on the other hand, the latter can be used if

one is not interested in computational devices, but only in foreseeing (as a first approximation) that some functions or relations (definable in the frame of first order theories) are (in principle) computable, or in guaranteeing that some expansions of given "intended models" satisfy some general requirements (having a character more semantical than syntactical). In this sense, a notion such as constructiveness (as defined by DP and EDP) turns out to be, according to the cases, undercharacterized or overcharacterized: subconstructiveness and semiconstructiveness are sufficient for ADT specification and extension. So, in the following, "constructive" provability will be provability in a system corresponding to any of the above notions of constructiveness.

2.5. CONSTRUCTIVENESS AND INDUCTION

Constructive logical proofs can be used to get expansion results. For example, let T be a (atomically complete) theory completely formalizing an ADT and let $T + L$ be a semiconstructive system. Let N be a set of new axioms characterizing a new relation symbol $r(x)$. In general, $T + N + L$ is no longer semiconstructive, or $T + N$ has no longer a reachable model. But suppose that $T + N$ has a reachable model, that $T + N + L$ is semiconstructive and that :

$$T + N + L \vdash \forall x(r(x) \vee \neg r(x)) ;$$

then, for every closed term t, $T + N + L \vdash r(t) \vee \neg r(t)$, and, by semiconstructiveness, $T + N + CL \vdash r(t)$ or $T + N + CL \vdash \neg r(t)$. Hence, $T + N$ is atomically complete and completely formalizes an ADT which is an expansion of the one formalized by T. This shows how semiconstructiveness (*a fortiori* constructiveness) can be used to state expansion results for ADT's. Our methodology is based on *sufficient conditions*, to be satisfied by the new axioms N in order that $T + N + L$ be, in turn, constructive (or semiconstructive), still preserving reachability. In the expansion results for parametric APC's, the notion of constructive (or semiconstructive) system will be replaced by the one of *virtually constructive* (or *virtually semiconstructive*) system. Intuitively, a virtually constructive (semiconstructive) system is a parametric system $T(X) + L$ (X being a list of parameters) which becomes constructive after correct instantiations of the parameters X.

Since we are interested in constructive proofs of universally quantified formulas such as $\forall x(r(x) \vee \neg r(x))$, *induction* becomes a relevant tool which *can* be used in our approach. Indeed, various kinds of induction schemes, as well as descending chain principles based on well founded orderings, can be used in a large class of ADT-theories T and pseudo logics L, preserving both the existence of an isoinitial model and the constructiveness or semiconstructiveness of $T + L$. Let us give an example.

Induction may be related to reachability. For example, let us consider lists with elements of sort EL. Every list can be built up in the usual way, starting from the constant $nil : LIST$, by the operation $| : EL \times LIST \rightarrow LIST$ (i.e., the carrier of the sort $LIST$ of lists is reachable by nil and $|$). Suppose we need a proof of a formula $\forall x, y H(x, y)$, with $x, y : LIST$. Using a natural-like deduction formalism (see Prawitz (1963)), we may prove the open formula $H(x, y)$, for example, by the following induction rule:

$$\frac{\begin{array}{cccc} & (H(nil, j)) & (H(i, nil) & (H(i, j)) \\ \Pi B & \Pi S_1 & \Pi S_2 & \Pi S_3 \\ H(nil, nil) & H(nil, b|j) & H(a|i, nil) & H(a|i, b|j) \end{array}}{H(x, y)}$$

with a *basis* ΠB and three *induction steps* ΠS_1, ΠS_2, ΠS_3. Parentheses in $(H(nil, j))$, $(H(i, nil))$, $(H(i, j))$ mean that the parenthesized formulas correspond to discharged assumptions. The soundness is based on the following facts:

- every couple (x, y) of lists is of one of the forms (nil, nil), $(nil, b|j)$, $(a|i, nil)$, $(a|i, b|j)$; we will say that $\{(nil, nil), (nil, b|j), (a|i, nil), (a|i, b|j)\}$ is a *cover set* for the couples of lists;
- in the induction steps, the complexity of the couples of terms of sort LIST in the discharged assumptions is less than the one of the corresponding terms in the consequences.

This is only an example of an induction rule based on a cover set, involving couples of variables (the variables x and y of the formula $H(x, y)$); other induction rules of this kind can be defined, involving single variables, triples of variables, and so on. For every cover set there are corresponding *(cover set) induction rules* (in general an induction rule working with one variable suffices). Other rules, having an inductive character and based on complexity measures of the terms (more generally, of the n-tuples of terms) representing the elements of the reachable carriers, can be considered, such as descending chain principles (see, e.g., Bertoni *et al.* (1984)). Induction rules can be used also for parametric theories, as we will see. They can be given also in the form of axiom-schemes, we will call *induction schemes.*

3. Operations on parametric signatures and theories

A *parametric signature* is a signature where some symbols and declarations are put into evidence as *parameters.* It will be indicated by $\Sigma(I)$, where I is the list of the parameters. Parametric declarations may contain only parametric sorts, and I determines a partition of Σ in two subsignatures Σ_I and Σ_D, where Σ_I contains *only the parameters* I and Σ_D contains the other symbols, called the *defined symbols* of $\Sigma(I)$.

A *parametric theory* is a theory $T_{\Sigma(I)} = C_{\Sigma_I} \cup Ax$, with a parametric signature $\Sigma(I) = \Sigma_I + \Sigma_D$ and two disjoint sets of axioms: the *constraints* C_{Σ_I} and the *proper (or internal) axioms* Ax. The constraints are the axioms containing *only parametric symbols* (i.e., C_{Σ_I} is a Σ_I-theory) while the proper axioms are the other ones (i.e. each of them contains at least a defined symbol). The proper axioms are intended to formalize the defined symbols of $\Sigma(I)$, while the constraints represent requirements to be satisfied by the actual parameters.

Let $T_{\Sigma(I)}$ be a parametric $\Sigma(I)$-theory, let I_Θ be a Θ-theory such that Θ and $\Sigma(I)$ are *disjoint signatures* (otherwise, a suitable renaming is made on $\Sigma(I)$), and let L be an intermediate pseudo logic (in the sense of Subsection 2.4). An *L-parameter passing from* $T_{\Sigma(I)}$ *to* I_Θ is a theory morphism

$$\sigma : C_{\Sigma_I} \rightarrow I_\Theta$$

such that, for every constraint K, $\sigma(K)$ is *provable* in $I_\Theta + L$. The *result* of the L-parameter passing operation is the theory $I_\Theta \cup \sigma(Ax)$.

The meaning of the resulting theory $I_\Theta \cup \sigma(Ax)$ is illustrated by the following commutative diagram, called a *parameter passing diagram* (as in Ehrig and Mahr (1985))

$$\begin{array}{ccc} C_{\Sigma_I} & \rightarrow_{\epsilon_1} & C_{\Sigma_I} \cup Ax \\ \downarrow_{\sigma} & & \downarrow_{\psi} \\ I_{\Theta} & \rightarrow_{\epsilon_2} & I_{\Theta} \cup \sigma(Ax) \end{array}$$

where σ replaces the parameters of Σ_I by symbols of Θ and ϵ_2 enriches the signature Θ by the defined symbols of Σ_D and gives rise to a new signature, we call the *enriched signature*. Correspondingly, every model of $I_{\Theta} \cup \sigma(Ax)$ is an expansion, by the defined symbols of Σ_D, of some model of I_{Θ}.

One of the uses of parameter passing is *instantiation* of a parametric theory $T_{\Sigma(I)}$ by a theory I_{Θ} *completely formalizing* an ADT A_{Θ}. Let $\Delta = \Theta + \sigma(\Sigma_D)$ be the enriched signature. Our aim is to obtain a *correct instantiation*, in the following sense:

The result of the L-parameter passing is *correct* if $I_{\Theta} \cup \sigma(Ax)$ completely formalizes an ADT, B_{Δ}, which is an ϵ_2-expansion of A_{Θ}. In this case we say that the L-parameter passing is a *correct L-instantiation* of $T_{\Sigma(I)}$ by I_{Θ} and that the resulting theory is a *L-instance of $T_{\Sigma(I)}$*. We will call L the instantiation (pseudo) logic.

Thus, a correct L-instantiation enriches by new sorts, functions and relations (interpreting the defined symbols of Σ_D) the ADT formalized by I_{Θ} and builds up the theory $I_{\Theta} \cup \sigma(Ax)$ formalizing the enriched ADT.

The first result obtained by our constructive methods is just a correctness result, proved in the next section. This result is based on the following notion:

Let P be an intermediate pseudo logic; we say that the morphism ϵ_1 (see the above diagram) *introduces P-defined symbols* if suitable formulas associated with the defined symbols are provable in $(C_{\Sigma_I} \cup Ax) + P$.

For instance, the formula associated with a relation $r(x)$ is $\forall x(r(x) \vee \neg r(x))$. Intuitively, this formula expresses the computability of $r(x)$. Henceforth, the formulas associated with the P-defined symbols will be called *computability formulas* and the P-defined symbols will be called also *P-computable* symbols. In Section 4 we will prove, together with other results, that if (under appropriate hypotheses) ϵ_1 introduces IKA-computable symbols, then every CL-instantiation is correct (with IKA and CL defined as in Section 2.4). We remark that here IKA-provability is required at the parametric level, while for instantiations classical provability suffices, i.e., classical and constructive methods can be mixed.

Morphisms introducing P-computable symbols are also the basis of the *P-class expansion* operation. A P-class expansion is a theory morphism:

$$\epsilon : T_{\Sigma(I)} \rightarrow T_{\Sigma(I)} \cup N_{\Delta}$$

introducing P-computable symbols but *no new constraints*. Under appropriate conditions and instantiation (pseudo) logics L, ϵ works as a *simultaneous expansion* of all the L-instances of the APC $T_{\Sigma(I)}$, automatically obtained by adding (the σ-translation of) N_{Δ} to the axioms of the L-instances (in the cases considered in this paper, P will be IKA and L will be CL).

Another way of building P-class expansions is *APC-P-composition*. It is a P-parameter passing, where (see the parameter passing diagram) $I_{\Theta} = I_{\Theta_I} \cup Ax'$ is a parametric theory (with I_{Θ_I} as the set of the constraints and Ax' as the set of the proper axioms) and σ replaces the parameters of C_{Σ_I} by defined symbols of I_{Θ}. The resulting theory is $I_{\Theta_I} \cup Ax' \cup \sigma(Ax)$, which is a P-class expansion of I_{Θ}. This expansion introduces

P-computable symbols, since, for every $K \in C_{\Sigma_I}$, $\sigma(K)$ is provable in $I_\Theta + P$ (as required by our definition of P-parameter passing). APC-P- composition is useful to compose APC's into bigger APC's in a sound way. IKA-class expansions and APC-IKA-compositions will be introduced in Section 4. APC-IKA-composition will be used in the last section to define *instantiations by IKA-recurrence.*

Finally, we consider operations introducing *L-subclasses.* The first one is *partial L-instantiation*, where some parameters are not substituted by σ and the corresponding constraints are not proved. In this case, the resulting theory is $C'_{\Delta_I} \cup I_\Theta \cup \sigma(Ax)$, where $\Delta_I \subseteq \Sigma_I$ are the non-instantiated parameters and $C'_{\Delta_I} \subseteq C_{\Sigma_I}$ are the related (unproved) constraints. The result of a partial L-instantiation represents a *L-subclass*, i.e., a class whose L-instances are L-instances of the parent class. An L-subclass can be introduced also by *parameter enlargements*, where new constraints and parameters are added. If only constraints are added, we obtain a L-subclass. Otherwise, we obtain a *L-pseudo subclass* with the following properties: the reducts of the isoinitial models of the L-instances of a L-pseudo subclass are isoinitial models of L-instances of the parent class.

L-subclasses *inherit* all the properties provable in the parent class. But we have a more interesting case, where also P-class expansions are inherited. We are interested in this kind of inheritance, we call *P-expansion L-inheritance*, where P is the logic used in expansions and L the one used in L-instantiations and in L-subclasses. The following diagram illustrates P-expansion L-inheritance for complete L-instantiations:

$$\begin{array}{ccccc} C_{\Sigma_I} & \rightarrow_{\epsilon_1} & C_{\Sigma_I} \cup Ax & \rightarrow_{\epsilon_3} & C_{\Sigma_I} \cup Ax \cup N_\Delta \\ / \downarrow_\sigma \setminus & & / \downarrow_{\psi_1} \setminus & & / \downarrow_{\psi_2} \setminus \\ I_\Theta & \rightarrow_{\epsilon_2} & I_\Theta \cup \sigma(Ax) & \rightarrow_{\epsilon_4} & I_\Theta \cup \sigma(Ax \cup N_\Delta) \end{array}$$

P-expansion L-inheritance holds if, for every L-instantiation $\sigma : C_{\Sigma_I} \rightarrow I_\Theta$, the expanded theory $I_\Theta \cup \sigma(Ax \cup N_\Delta)$ completely formalizes an isoinitial model which is an ϵ_4-expansion of the isoinitial model of $I_\Theta \cup \sigma(Ax)$. P-expansion L-inheritance for L-subclasses follows in an obvious way. As said above, in the following we will be mainly interested in IKA-expansion CL-inheritance, we will call simply *expansion inheritance.* The general study of P-expansion L-inheritance for P and L different from IKA and CL, may be interesting, as briefly pointed out in Section 4.

The notion of isoinitiality, on which the above definitions depend, is based on classical Model Theory. On the other hand, we have repeatedly quoted constructive provability. Now, let us better explain how constructiveness is used. In line with the results of the next section, in general a *complete* formalization T^* of an ADT can be fully captured in terms of the constructiveness of a system $T^* + P$ (this is strictly related to the *recursiveness* of ADT's, as explained in the previous Sections). On the other hand, the fact that T incompletely formalizes an ADT (i.e., T formalizes an APC) corresponds to a lack of constructiveness to be completed (by instantiations): in this sense, the *constraints* are used as an *anticipation of a "constructive content"* (i.e., they are a *constructiveness requirement*) *to be further provided by instantiations.* Thus, a theory T formalizing an APC is to be seen (under an appropriate intermediate pseudo logic P) as *virtually, but not actually, constructive.* Correspondingly, morphisms $\epsilon_1, \epsilon_3, \ldots$ introduce *virtually, but not actually, recursive* relations and functions at the APC-level; after a L-instantiation, the corresponding morphisms $\epsilon_2, \epsilon_4, \ldots$ introduce *actually recursive* relations and functions. The *constraints* and the *constructive provability of the computability formulas* play a central role at two levels. At the class level, the constraints K are used to constructively

prove in P the computability formulas (i.e. to state the virtual computability of the defined symbols). At the instantiation level, the L-provability of the translations $\sigma(K)$ of the constraints is used whenever it can guarantee that the actual parameters satisfy the computability assumptions made at the class level. In the cases considered in Section 4, where the constraints have a *particular form*, CL-provability of the constraints suffices.

We conclude this section by introducing *APC-expressions*, i.e., the language we use to define APC's and operations on them. *Unless otherwise stated*, the (constructive) intermediate pseudo logic involved in our APC-expressions and in our examples will be IKA for expansions and APC-compositions, while CL will be used for instantiations and subclasses. Then "expansion" stands for "IKA-expansion", "instantiation" for "CL-instantiation", and so on.

We indicate the APC of a parametric theory T by an expression such as:

Class $T(D_1, \ldots, D_k; I_1, \ldots, I_h)$;
signature;
axioms;
theorems
End Class.

$T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ is the *heading expression*, $D_1, \ldots, D_k$ are defined symbols, and $I_1, \ldots, I_h$ are the parameters. We will call $I_1, \ldots, I_h$ *input parameters* and $D_1, \ldots, D_k$ *defined parameters*. The defined parameters are the defined symbols which can be renamed in APC-composition (renaming will be needed in instantiation by recurrence), while the defined symbols different from $D_1, \ldots, D_k$ are considered as standard names (it is merely a question of convenience to use always the same names, e.g., for basic operations on lists). In the heading, if no ambiguity can arise, instead of a declaration of a function, relation or constant we will list only the corresponding function, relation or constant symbol. The axioms are divided in *internal (or proper) axioms* and *constraints*. In the theorems, constructive proofs of computability formulas are listed as *computability theorems*.

The result of a (possibly partial) *L-parameter passing from a parametric theory* $T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ *to a theory* T' is described by a *heading expression* such as:

$$T(R_1, \ldots, R_k; A_1 :: T', \ldots, A_h :: T')$$

where $R_1, \ldots, R_k$ are the (possibly) new names of the defined parameters and $A_1, \ldots, A_h$ are the actual parameters (i.e. the symbols of T' replacing the input parameters). The L-provability of the (σ-translation of the) constraints is checked; if it holds, then the resulting theory is built up and the constraints are deleted, according to the definition of L-parameter passing. In *partial parameter passing*, if some input parameter is not replaced, then the corresponding constraints are neither deleted nor checked and the qualifier $:: T'$ is omitted.

The language to define class expansions and parameter enlargements will be explained later. In order to clarify our explanation of APC-expressions, let us consider an example; in this example we introduce also some further details on APC-expressions.

Class $List(LIST; X)$;
Signature:

sorts: X, $LIST$;
consts: $nil : LIST$;
functs: $..|.. : X \times LIST \rightarrow LIST$
rels: $=: X \times X$;
$=: LIST \times LIST$

Internal axioms

Constructors internal axioms (see the constructors theories, in Section 2.3):

(inj1) $\forall x, y : \neg nil = x|y$
(inj2) $\forall x, y, u, v : (x|y = u|v \rightarrow x = u \wedge y = v)$

Induction rules, e.g. (see also Section 2.5):

$$\frac{H(nil) \qquad \begin{matrix}(H(y))\\ H(x|y)\end{matrix}}{H(z)}$$

Constraints:

Constructors constraints:

$\forall x, y : X(x = y \vee \neg x = y)$

Computability theorems:

$\forall x, y : LIST(x = y \vee \neg x = y)$

[Decidability of $=: LIST \times LIST$; proof automatically guaranteed (see points c), d) below)]

End Class.

Some conventions and remarks.

a) The *identity symbol* $=: s \times s$ is always *implicitly included* for every sort symbol s; $=$ on input parametric sorts is assumed to be an implicit input parameter (in our example, $=: X \times X$). Also the identity axioms are always implicitly included. Of course, if a parametric sort is instantiated, then also the related identity becomes instantiated. So, in our example we have:
input parameters: X, $=: X \times X$; defined parameters: $LIST$; other defined symbols: nil, $|$, and $=: LIST \times LIST$.

b) In our example, the *constructors internal axioms* formalize the fact that we construct in a unique way any list starting from nil by the operation $|$; (inj1), (inj2) are the injectivity axioms considered in Section 2.3.

c) The *computability formula* $\forall x, y : LIST(x = y \vee \neg x = y)$ can be constructively proved as follows, using the cover set induction of Section 2.5, the identity and injectivity axioms, and the *constraint* on $=: X \times X$:

$$\frac{\dfrac{\dfrac{}{nil = nil}}{D(nil = nil)} \quad \dfrac{\dfrac{}{\neg a|I = nil}}{D(a|I = nil)} \quad \dfrac{\dfrac{}{\neg nil = b|J}}{D(nil = b|J)} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg I = J)}}{\neg a|I = b|J}}{(D(I = J))} \quad \dfrac{\dfrac{\dfrac{}{(a = b)}}{a|I = b|J}}{D(a|I = b|J)} \quad \dfrac{\dfrac{\dfrac{}{(I = J)}}{(\neg a = b)}}{D(a|I = b|J)}}{D(a = b)} \quad \dfrac{\dfrac{\dfrac{}{(\neg a = b)}}{(\neg a|I = b|J)}}{D(a|I = b|J)}}{D(a|I = b|J)}}{D(x = y)}$$

where, to save space, we use $D(A)$ to abbreviate $A \vee \neg A$. Notice that $nil = nil$, $a = b \wedge I = J \rightarrow a|I = b|J$ follow from the identity rules, while $\neg a|I = nil$, $\neg nil = b|J$, $\neg I = J \rightarrow \neg a|I = b|J$, $\neg a = b \rightarrow \neg a|I = b|J$ are *proved* by the injectivity axioms, and $D(a = b)$, i.e. $a = b \vee \neg a = b$ (with $a, b : X$) is *proved* by the constraint $\forall x, y : X(x = y \vee \neg x = y)$. Facts of this kind *automatically* hold in presence of sorts whose carriers are generated by constructors; also, the "constructors internal axioms", the "constructors constraints" and proofs such as the above can be *automatically generated.*

d) In our APC-expressions, injectivity axioms, induction rules, constraints and decidability proofs such as the above are automatically generated by the *statement:*

"LIST constructed by *nil*, |";

so, the above class definition will be provided by the following APC-definition:

Class $List(LIST; X)$;
Signature:
sorts: X, $LIST$;
consts: $nil : LIST$;
functs: $..|.. : X \times LIST \rightarrow LIST$
rels: $=: X \times X$;
$=: LIST \times LIST$

Internal axioms
LIST constructed by |, *nil*

Constraints, Computability theorems: IMPLICIT
End Class.

We point out that the above point c) *does not imply* the constructiveness of the system $List(LIST; X) + IKA$, but it implies the following instantiation property: let T^* be a theory completely formalizing an ADT and satisfying appropriate conditions such as the hypotheses of Theorem 4.2 of the next section, assuring the constructiveness of the system $T^* + IKA$; suppose that X is instantiated by a sort s of T^* and $=: X \times X$ on $=: s \times s$; then the instantiation gives rise to a constructive system (by the results of the next section) which completely formalizes an ADT. For example, let us consider the following theory NAT formalizing the ADT of the natural numbers:

ADT NAT;
Signature:
sorts: N;
consts: $0 : N$;
functs: $succ : N \rightarrow N$
rels: $=: N \times N$;
Axioms
N constructed by 0, *succ*
End ADT.

We can instantiate the above class, as follows:

Instance $ListNat = List(ListOfN; N :: NAT)$;

The instance $ListNat$ consists of the following axioms (in our APC-expressions, "Instance ..." works as a statement which automatically generates an instance):

a) the identity axioms on the polimorphic $=$;
b) the axioms and rules coming from NAT:

(injN1) $\forall x : \neg 0 = succ(x)$

(injN2) $\forall x, y : (succ(x) = succ(y) \rightarrow x = y)$

(indN) $$\frac{H(0) \quad \overset{(H(x))}{H(succ(x))}}{H(z)}$$

and the other induction rules, we omit

c) the instantiated internal axioms of the instantiated class:

(injL1) $\forall x, y : \neg nil = x|y$

(injL2) $\forall x, y, u, v : (x|y = u|v \rightarrow x = u \wedge y = v)$

(indListOfN) $$\frac{H(nil) \quad \overset{(H(y))}{H(x|y)}}{H(z)}$$

and the other instantiated induction rules

4. Some results

4.1. THEORIES COMPLETELY FORMALIZING ADT'S

First of all, as an almost immediate consequence of Theorem 2.1, we state a sufficient condition in order that a theory T completely formalizes an ADT:

THEOREM 4.1. *Let T and L be, respectively, a theory and a psuedo logic such that the following conditions are satisfied:*

1) T has a reachable model; 2) $S = T + L$ is a semiconstructive formal system; 3) for every relation r of the signature of T, $S \vdash \forall \underline{x}(r(\underline{x}) \vee \neg r(\underline{x}))$.

Then T completely formalizes an ADT, i.e., any reachable model of T is an isoinitial model of T.

The above sufficient condition can be made a necessary and sufficient condition for suitable theories, under suitable pseudo logics L.

This aspect involves, on the other hand, sufficient conditions in order that a theory be semiconstructive (constructive) under a pseudo logic L and requires some definitions.

- A $\forall$-*formula* is any formula of the kind $\forall \underline{x} H$, where H is a quantifier free formula (by notations such as $\forall \underline{x}$ and $\exists \underline{x}$ we indicate possibly empty lists of quantifiers).
  A $\exists$-*formula* is any formula of the kind $\exists \underline{x} H$, where H is a quantifier free formula.
  A $\forall\exists$-*formula* is any formula of the kind $\forall \underline{x} \exists \underline{y} H$, where H is a quantifier free formula.
  A $\forall\exists\neg$-*formula* is inductively so defined: every $\forall\exists$-formula is a $\forall\exists\neg$-formula; every

negated formula $\neg H$ (for any H) is a $\forall\exists\neg$-formula; if A and B are $\forall\exists\neg$-formulas and C is any formula, $A \wedge B$, $C \rightarrow A$ and $\forall \underline{x} A$ are $\forall\exists\neg$-formulas.
A $\forall\neg$-*formula* is inductively defined as a $\forall\exists\neg$-formula, taking in the basic clause the $\forall$-formulas instead of the $\forall\exists$-formulas.
An *Harrop formula* (see Troelstra (1973)), which we call also an *elementary formula*, is inductively so defined: every atomic or negated formula is elementary; if A and B are elementary and C is any formula, then $A \wedge B$, $C \rightarrow A$ and $\forall \underline{x} A$ are elementary formulas.

- A $\forall\exists\neg$-*theory* is any theory T such that: 1) T (possibly) contains induction schemes (also in the form of descending chain principles) without any restriction (i.e., the instances of the schemes correspond to arbitrary formulas); 2) all the formulas (axioms) of T which are not instances of induction-schemes are $\forall\exists\neg$-formulas.
$\forall\exists - \forall\neg$-*theories*, $\forall\neg$-*theories, and elementary theories* are defined as the $\forall\exists\neg$-theories, in the first case taking in clause 2) formulas which are either $\forall\exists$-formulas or $\forall\neg$-formulas (instead of arbitrary $\forall\exists\neg$-formulas), in the second case taking in clause 2) only $\forall\neg$-formulas and in the third case taking in clause 2) only elementary formulas. Of course, any elementary theory is a $\forall\neg$-theory, a $\forall\neg$-theory is a $\forall\exists-\forall\neg$-theory and any $\forall\exists - \forall\neg$-theory is a $\forall\exists\neg$-theory.

Taking into account the logic IKA, we can prove the following theorem (see, e.g., Miglioli *et al.* (1989); as far as the logic IKA is involved, an outline of the proof is given in Bertoni *et al.* (1984), where IKA is called CON):

THEOREM 4.2. *Let T be any atomically complete $\forall\exists\neg$-theory with a reachable model. Then T + IKA is a constructive formal system.*

In Miglioli *et al.* (1989) various pseudo logics L stronger than IKA are shown, which give rise to semiconstructive or constructive systems $T + L$, where the semiconstructiveness or the constructiveness of a system $T + L$ depends both on T and on L. An idea of these results is given in Section 4.4. Here we are not interested in exploring the possibilities involved in the use of larger and larger (semi) constructive systems and in looking for examples which can be better treated under some pseudo logics instead of some others. We are at the very beginning of such an exploration; in this sense, the pseudo logic IKA will be sufficient for the purposes of the present paper.

Now, we come back to the characterization of theories where the sufficient conditions of Theorem 4.1 can be made also necessary conditions. The following definitions are in order.

- Let Σ and $r : \underline{a}$ be, respectively, a (many sorted overloaded) signature and a relation declaration of Σ (not necessarily an input relation declaration). By the *canonical constraint associated with* $r : \underline{a}$, denoted by $cc(r : \underline{a})$, we will mean the formula $\forall \underline{x}(r(\underline{x}) \vee \neg r(\underline{x}))$. By the set of the canonical constraints associated with Σ, denoted $cc(\Sigma)$, we will mean $\{cc(R) | R \text{ is a relation declaration of } \Sigma\}$.
- Let T be any theory and let Σ be the signature of T. By the canonical extension of T, denoted by $cc(T)$, we will mean the theory $T \cup cc(\Sigma)$.

Of course, the formulas of $cc(\Sigma)$ are quite irrelevant from the classical point of view, i.e., T and $cc(T)$ are classically equivalent and, since the new axioms do not affect the class

of models of T, T completely formalizes an ADT iff $cc(T)$ does. On the other hand, these formulas are quite relevant from a constructive point of view and will be the basis of our methodology. In this line, we remark that canonical constraints $cc(R)$ are $\forall$-formulas, hence $\forall\exists\neg$-formulas. Thus, we can combine Theorem's 4.1 and 4.2 and state, in "purely constructive terms", a necessary and sufficient condition in order that a class of relevant theories completely formalizes an ADT:

THEOREM 4.3. *Let T be any $\forall\exists\neg$-theory. Then T completely formalizes an ADT iff the two following conditions are satisfied: 1) T (and hence $cc(T)$) has a reachable model; 2) $cc(T)$ + IKA is a constructive formal system.*

REMARKS.

- Since the presence of the formulas $\forall\underline{x}(r(\underline{x}) \vee \neg r(\underline{x}))$ allows to prove intuitionistically the rule (A) (i.e., $\neg\neg A \rightarrow A$ for any atomic A), (A) becomes redundant in $cc(T)$ + IKA.
- Theorem. 4.3 can be restated for $\forall\neg$-theories as a necessary and sufficient condition for atomic completeness only: a $\forall\neg$-theory T is atomically complete iff $cc(T)$ + IKA is constructive.

4.2. THEORIES FORMALIZING APC'S

If a ($\forall\exists\neg$-theory) T does not completely formalize an ADT, then $cc(T)$+IKA *may well be non-constructive*. However, there are cases where $cc(T)$ can be looked at as a theory to be extended, in such a way that (the extension of) $cc(T)$ becomes a theory with a reachable model and gives rise (with IKA or some other appropriate pseudo logic) to a constructive system, thus automatically becoming a theory completely formalizing an ADT. This is the idea involved in our methodology, which we are going to explain.

First of all, we come to the notion of an APC. From a general, semantical point of view (whether or not we are able to state that a given parametric theory is indeed an APC) such a notion is explained as follows:

Abstract parametric classes. Let $T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ be a (classically) † consistent parametric theory, let L be an intermediate pseudo logic and let $\mathcal{C}$ be a class of theories. We say that $T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ *formalizes a L-APC with respect to $\mathcal{C}$* iff the following condition is satisfied:

let T' be any theory of $\mathcal{C}$ such that T' completely formalizes an ADT and the L-parameter passing $T(R_1, \ldots, R_k; A_1 :: T', \ldots, A_h :: T')$ is consistent: then $T(R_1, \ldots, R_k; A_1 :: T', \ldots, A_h :: T')$ is a *correct* L-instantiation, i.e. it completely formalizes an ADT which is an expansion of the one formalized by T'.

If $\mathcal{C}$ is the class of all the theories and T formalizes a CL-APC (CL is classical logic) with respect to $\mathcal{C}$, we simply say that T is an *absolute* APC.

Unless otherwise stated, we will consider only CL-instantiations (simply called instantiations), CL-APC's (simply called APC's) and absolute APC's. The following definitions

† Henceforth, by "consistent" we will mean "classically consistent"

are needed to provide reasonable sufficient conditions in order that a parametric theory be an APC.

- Let $T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ be a consistent parametric theory and let $\mathcal{C}$ be a class of theories. We say that $T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ is *reachable with respect to* $\mathcal{C}$ iff the following condition is satisfied: let $T' \in \mathcal{C}$ be a theory such that the CL-parameter passing $T(R_1, \ldots, R_k; A_1 :: T', \ldots, A_h :: T')$ is consistent and T' has a reachable model: then $T(R_1, \ldots, R_k; A_1 :: T', \ldots, A_h :: T')$ has a reachable model.
  If $\mathcal{C}$ is the class of all theories and T is reachable with respect to $\mathcal{C}$, we simply say that T is *parametrically reachable.*
- Let T be a consistent parametric theory. We say that T is *regular* iff the following conditions hold:
  1) for every input relation declaration $r : \underline{a}$ of the signature of T, the canonical constraint $cc(r : \underline{a})$ belongs to T;
  2) the other formulas of T are either elementary formulas or instances of induction schemes.
- We say that T is *IKA-virtually constructive* (IKA-v.c.) iff the following condition holds: for every defined relation declaration $r : \underline{a}$ of the signature of T, $T + \text{IKA} \vdash cc(r : \underline{a})$.

Now, as an easy consequence of Theorem 4.3, one can prove: if $\mathcal{C}$ is the class of the $\forall\exists\neg$-theories and T is any regular and IKA-v.c. theory which is parametrically reachable with respect to $\mathcal{C}$, then T is an APC with respect to $\mathcal{C}$.

This result, however, can be improved into the following one, which requires a direct proof:

THEOREM 4.4. *Let T be any parametric theory such that T is regular, parametrically reachable and IKA-v.c.: then T is an absolute APC.*

PROOF. To prove Theorem 4.4, one has to consider arbitrary extensions Σ' of the signature Σ of T obtained by introducing an arbitrary non-empty set of new constants for every relevant sort, where a sort is relevant if it is an input parameter. Any extension Σ' of Σ automatically gives rise to a set τ' of closed terms which extends the set τ related to Σ: the relevant terms of τ' will be the ones whose sort is relevant in the above sense. Now, given any extension from Σ to Σ', one considers arbitrary extensions $T' = T \cup T''$, where T'' satisfies the following properties:
1) T'' *is consistent with* T;
2) if $r : \underline{a}$ is any input relation declaration and $\underline{t}$ is a sequence of (relevant) terms of sorts $\underline{a}$, then $r(\underline{t}) \in T''$ or $\neg r(\underline{t}) \in T''$, and these are the only formulas of T''.
Taking into account such extensions T' of T, Theorem 4.4 easily follows from the following fact:
(*) For any T', $T' + \text{IKA}$ is constructive.
The proof of (*) can be carried out using the method outlined in Bertoni *et al.* (1984) to prove that, for every $\forall\exists\neg$-theory T completely formalizing an ADT, $T + \text{IKA}$ is constructive (this method is based on the collection technique explained in Miglioli and Ornaghi (1981)). □

REMARK. The input parameters of a theory T satisfying the hypotheses of Theorem

4.4 may be replaced by defined symbols of a theory T' satisfying the same hypotheses as T (i.e., T' is a regular, parametrically reachable and IKA-v.c. theory), using an IKA-parameter passing. One easily proves that the resulting APC $T(R_1, ..., R_k; A_1 :: T', ..., A_h :: T')$ is an absolute APC (provided that it is consistent). Then the class of the APC's considered in Theorem 4.4 is *closed under APC-IKA-composition.*

To complete our effective characterization of APC's, i.e., to provide significant sufficient conditions in order that T be an APC, we must give some sufficient conditions for parametric reachability (note that Theorem 4.4 essentially characterizes the relations of T, not its functions). Later we will provide results to extend APC's into APC's, where the extensions allow also the introduction of new functions. Here we give a simple characterization of the parametric reachability of a particular, but relevant class of APC's (this class can be seen as a starting point, from which larger classes can be obtained by extensions).

- We say that a term t of a parametric signature is a *parametric term* if every variable occurring in t (if any) has a sort s which is an input parameter.
- Let Σ be a parametric signature such that, for any defined sort s, the set T_s of the parametric terms of sort s is not empty. By the *parametric theory of* Σ, *denoted* $PT(\Sigma)$, we mean the regular parametric theory $T = T_1 \cup T_2 \cup T_3$ so defined:
  1) T_1 is the set $Inj(C)$ of the *injectivity axioms* of the set C of the defined constant and function declarations of Σ;
  2) T_2 contains all the instances of all the *induction schemes*, where there are induction schemes for every defined sort (instead of the induction schemes one can use the corresponding rules, e.g., in the formalism of natural calculus).
  3) T_3 is the set of all the canonical constraints $cc(r : \underline{a})$, for every input relation declaration $r : \underline{a}$ of Σ.

Given a parametric theory $PT(\Sigma)$, we call *constructors* the constant and function symbols involved in the injectivity axioms, and we also call $PT(\Sigma)$ a *parametric constructors theory*; also, if Σ contains relation declarations *only* for overloaded identity, and function and constant declarations *only* for constructors, then we call $PT(\Sigma)$ a *pure parametric constructors theory*.

The usual identity axioms are implicitly assumed in the logics used to make deductions from $PT(\Sigma)$, but one can, alternatively, include them in $PT(\Sigma)$. One can also include in $PT(\Sigma)$ (without affecting its properties relevant for our results):
a) a partial ordering relation $\preceq_s$ (axiomatized in the obvious way) for any defined sort s, such that, for any two closed terms t_1, t_2 of sort s, $t_1 \preceq_s t_2$ iff t_1 is a subterm of t_2;
b) a descending chain principle for every defined sort s based on $\preceq_s$ (Bertoni *et al.*(1984)).

One can easily prove that $PT(\Sigma)$ is parametrically reachable. Also, $PT(\Sigma)$ is trivially a regular theory. Finally, one can show that there is an induction proof of

$$PT(\Sigma) + \mathrm{IKA} \vdash \forall x, y : s(x = y \vee \neg x = y)$$

for every defined sort s. Indeed, it is not difficult to see that a proof such as the one shown for lists in Section 3 can be obtained in the general case, using the injectivity axioms and the canonical constraints on parametric (overloaded) identities. Thus, one has:

THEOREM 4.5. *For every parametric* Σ, $PT(\Sigma)$ *is a regular parametrically reachable theory. Also, if* $PT(\Sigma)$ *is a pure parametric constructors theory, then* $PT(\Sigma)$ *is IKA-v.c. (it is an absolute APC).*

We can extend the notion of regular parametric theory into the more general notion of *almost regular* parametric theory T. Such a theory has the form $T = T' \cup T''$, where T' is a regular parametric theory and T'' is a (possibly empty) set of $\forall\exists\neg$-*formulas* containing only input parameters: we call them *non-canonical* constraints.

The non-canonical constraints may be used to prove in IKA the canonical constraints related to the defined relations of a theory. In this sense, *they strengthen the power* of APC formalizations (we will present examples with these constraints). Unfortunately, non-canonical constraints, differently from the canonical ones, are not necessarily classically valid. In this case a theory morphism σ, CL-instantiating the input parameters, must translate every constraint K into a $\sigma(K)$ *classically provable* in the theory T^* (completely formalizing an ADT) used to instantiate such a theory T. If this fact is satisfied, we will say that T^* *agrees with* T. Thus, for almost regular parametric theories, Theorem. 4.4 must be rewritten as follows.

THEOREM 4.6. *Let* T *be any parametric theory such that* T *is almost regular, IKA-v.c. and parametrically reachable with respect to the class* C *of all the theories* T' *which agree with* T. *Then* T *is an APC with respect to* C.

Also the notion of non-canonical constraint may be unnecessarily restrictive in some context. For instance, if a pseudo logic L is used in L-instantiations and a pseudo logic P is used to introduce P-computable symbols, one can use as constraints on input parameters arbitrary formulas F, provided that the following condition is satisfied:

> Let T^{**} be the result of a L-instantiation (by some ADT-theory T^*); then $T^{**} + L + P$ can be extended in a constructive system.

In practice, the condition $L \subseteq P$ is sufficient (a situation different from the one of Theorem 4.6, where $P = IKA \subsetneq L = CL$).

Now we briefly explain how the above results are used in our methodology. In the following we call *constraints* the constraints on the input relation declarations and *computability formulas*, or also *defined constraints*, the canonical constraints of the defined relation declarations. Using Theorem 4.5, our methodology defines APC's formalized as pure constructors theories by APC-definitions such as:

Class $T(D_1, .., D_n; X_1, ..., X_m)$;
signature:
sorts: $D_1, \ldots, D_n, X_1, \ldots, X_m$;
consts: ...;
functs: ...;
internal axioms: D_1 constructed by K_1;...; D_n constructed by K_n;
constraints: IMPLICIT;
computability proofs: IMPLICIT;
End Class.

where:

- K_i is the set of the constructors of sort D_i $(1 \leq i \leq n)$;
- overloaded identity is implicit;
- the constraints are implicit, because we have a regular theory;
- the computability proofs, i.e., the proofs of the defined constraints, are implicit, since the only defined relation is (overloaded) identity, and in a constructors theory the related defined constraints can be proved by induction.

We point out that in a pure constructors theory every constant or function declaration occurs in some K_i. If a constructors theory is not pure, so that it contains relation declarations different from identity, then the computability proofs for identity are still implicit, while we have to explicitly prove (in IKA) the ones related to the other defined relation declarations. If our theory is almost regular, the canonical constraints will be still implicit, but this is not the case for the non-canonical ones, which will be explicitly given.

Finally, if we have function or constant declarations which are not constructors (i.e. the involved theory is not a constructors theory), then we have also to explicitly prove (in IKA) the canonical constraints of the defined identity relations (however, in general also these constraints easily come from the induction rules). In many cases, the introduction of relations different from identity and of functions different from the constructors can be given in a simpler way by step by step extensions, as explained in the next subsection. Also, the expansion inheritance properties of extensions allow to propagate an extension to whole classes of instances or of subclasses.

4.3. EXTENSIONS OF APC'S

Now we show how one can extend an APC T into an APC T' (with new functions and relations, where the extension may be also non-conservative), in such a way that all the instances of T inherit the new functions of T' in the sense explained above, in Section 3. To do so, we need the following definition.

A formula A (in a language L_Σ) is said to be *basic* in L_Σ iff there is a $B \in L_\Sigma$ such that B is classically equivalent to A (i. e., $\emptyset + \mathrm{CL} \vdash A \leftrightarrow B$, where $\emptyset$ is the empty Σ-theory) and B is a $\forall$-formula.

Notice that two classically equivalent formulas may have quite different properties from a constructive point of view. Now we state our extension result:

THEOREM 4.7. *Let Σ be any parametric signature and let T be a parametric Σ-theory such that T is regular, parametrically reachable and IKA-v.c. (hence, T is an absolute APC). Then we have the two following extension results:*

(I) Let $r : \underline{a}$ be a new relation declaration. Let $\Sigma'_1 = \Sigma \cup \{r : \underline{a}\}$ with $r : \underline{a}$ a defined parameter of Σ'_1. Let $T'_1 \subseteq L_{\Sigma'_1}$ be a set of formulas such that, for every $H \in T'_1$, H is an elementary basic formula or H is an instance, in the extended language $L_{\Sigma'_1}$, of some induction scheme contained in T.

Let $T_1^{ext} = T \cup T'_1$ and let T_1^{ext} be consistent. Suppose that $T_1^{ext} + \mathrm{IKA} \vdash \forall x(r(x) \vee \neg r(x))$. Then T_1^{ext} is an absolute APC extending T in such a way that the following **class inheritance property** *(ip1) holds (where, using APC-expressions, we represent T as*

$T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ and T_1^{ext} as $T_1^{ext}(D_1, \ldots, D_k, r : \underline{a}; I_1, \ldots, I_h)$; note that T_1^{ext} has the same input parameters as T and has one defined parameter more than T, i.e., $r : \underline{a}$):

(ip1) if T^* is any theory completely formalizing an ADT and $T_1^{ext}(R_1, \ldots, R_k, r : \underline{a}; A_1 :: T^*, \ldots, A_h :: T^*)$ is consistent, then any ADT (isoinitial model) I completely formalized by $T(R_1, \ldots, R_k; A_1 :: T^*, \ldots, A_h :: T^*)$ can be expanded into an ADT (isoinitial model) I' completely formalized by $T_1^{ext}(R_1, \ldots, R_k, r : \underline{a}; A_1 :: T^*, \ldots, A_h :: T^*)$
(vice-versa, the restriction to the signature of $T(R_1, \ldots, R_k; A_1 :: T^*, \ldots, A_h :: T^*)$ of any ADT I' completely formalized by $T_1^{ext}(R_1, \ldots, R_k, r : \underline{a}; A_1 :: T^*, \ldots, A_h :: T^*)$ is an ADT completely formalized by $T(R_1, \ldots, R_k; A_1 :: T^*, \ldots, A_h :: T^*)$).

(II) Let $f : \underline{a} \to s$ be a new function declaration and $r : \underline{a}s$ be an associated relation declaration. Let $\Sigma_2' = \Sigma \cup \{r : \underline{a}s\}$, $\Sigma_2'' = \Sigma \cup \{r : \underline{a}s, f : \underline{a} \to s\}$, $r : \underline{a}s$ be a defined parameter of Σ_2' and of Σ_2'' and $f : \underline{a} \to s$ be a defined parameter of Σ_2''. Let $T_2' \subseteq L_{\Sigma_2'}$ be a set of formulas such that, for every $H \in T_2'$, H is an elementary basic formula or H is an instance, in the extended language $L_{\Sigma_2'}$, of some induction scheme contained in T.

Let $T \cup T_2'$ be consistent and let us suppose that $T \cup T_2' + \mathrm{IKA} \vdash \forall x \exists! y\, r(x, y)$. Let $T_2^{ext} = T \cup T_2' \cup \{\forall x\, r(x, f(x))\} \cup \{\mathrm{IND}''\}$, where IND'' is the set of all the instances corresponding to formulas of $L_{\Sigma_2''}$ of the induction schemes contained in $T \cup T_2'$. Then T_2^{ext} is an absolute APC and the following **class inheritance property** *(ip2) holds (we represent T as $T(D_1, \ldots, D_k; I_1, \ldots, I_h)$ and T_2^{ext} as $T_2^{ext}(D_1, \ldots, D_k, r : \underline{a}s, f : \underline{a} \to s; I_1, \ldots, I_h)$):*

(ip2) if T^* is any theory completely formalizing an ADT and $T(R_1, \ldots, R_k, r : \underline{a}s, f : \underline{a} \to s; A_1 :: T^*, \ldots, A_h :: T^*)$ is consistent, then any ADT (isoinitial model) I completely formalized by $T(R_1, \ldots, R_k; A_1 :: T^*, \ldots, A_h :: T^*)$ can be expanded into an ADT (isoinitial model) I' completely formalized by $T(R_1, \ldots, R_k, r : \underline{a}s, f : \underline{a} \to s; A_1 :: T^*, \ldots, A_h :: T^*)$ (vice-versa, the restriction to the signature of $T(R_1, \ldots, R_k; A_1 :: T^*, \ldots, A_h :: T^*)$ of any ADT I' completely formalized by $T(R_1, \ldots, R_k, r : \underline{a}s, f : \underline{a} \to s; A_1 :: T^*, \ldots, A_h :: T^*)$ is an ADT completely formalized by $T(R_1, \ldots, R_k; A_1 :: T^*, \ldots, A_h :: T^*)$).

PROOF. In order to prove Theorem 4.7, Point (I), one has, as a consequence of Theorem 4.4, that the theory T_1^{ext} is an absolute APC. Thus, one has only to show that the inheritance property (ip1) holds.

The proof of the latter fact can be reduced to the following one, which is a particular case of a theorem stated in Miglioli *et al.* (1989):
Let T^* be a theory completely formalizing an ADT. Let $T^{**} = T^* \cup T^{***}$, let Σ^* be the signature of T^*, and let $\Sigma^{**} = \Sigma^* \cup \{r : \underline{a}\}$ be the signature of T^{**}. Let all the formulas of T^{***} be elementary basic formulas or instances, corresponding to formulas of $L_{\Sigma^{**}}$, of induction schemes contained in T^*. Let $T^{**} + \mathrm{IKA}$ be constructive, and let $T^{**} + \mathrm{IKA} \vdash \forall x(r(x) \vee \neg r(x))$. Then any isoinitial model of T^* can be expanded into an isoinitial model of T^{**}.

In turn, the proof of this fact depends on the form of the formulas of T^{***}: one extends any isoinitial model I^* of T^* into a structure S for the language $L_{\Sigma^{**}}$, where the carriers and the old relations and functions are left unchanged, while the new relation r is such that $r(\underline{t})$ ($\underline{t}$ closed terms) holds in S iff $T^{**} + \mathrm{CL} \vdash r(\underline{t})$ (iff $T^{**} + \mathrm{IKA} \vdash r(\underline{t})$). Then one

shows that all the formulas of T^{***} (having basic or inductive instances) hold in S, from which one deduces that S is a model of T^{**}, hence an isoinitial model of T^{**}.

The proof of Theorem 4.7, Point (II), can be reduced to the proof of Theorem. 4.7, Point (I), since an IKA-proof of $\forall x \exists! y\, r(\underline{x}, y)$ allows to build up an IKA-proof (starting from the same theory) of $\forall \underline{x}, y\,(r(\underline{x}, y) \vee \neg r(\underline{x}, y))$. This fact, according to the above, allows to extend any isoinitial model I^* of T^* (with signature Σ^*) into an isoinitial model I^{**} of $T^{**} = T^* \cup T^{***}$ (with signature $\Sigma^{**} = \Sigma^* \cup \{r : \underline{a}s\}$), where the relation r is functional. Thus, it suffices to "rename" this relation r into a function f which does not affect the carriers of I^* (for the sake of conciseness, here we omit further details). □

Extension results similar to Theorem 4.7 or to the definitory extensions explained in the following remark can be stated, *mutatis mutandis*, for *almost regular theories* satisfying the hypotheses of Theorem 4.6.

Another way to extend an APC (which is in line with analogous results stated in Bertoni *et al.* (1984), Miglioli *et al.* (1989), allowing to obtain definitory – and therefore conservative – extensions of theories completely formalizing ADT's) is based on the following points (we omit the proof).

Conservative extension a) Let T, with signature Σ, be a regular, parametrically reachable and IKA-v.c. theory. Let $T + \mathrm{IKA} \vdash \forall x \exists! y A(\underline{x}, y)$, where A is any formula. Let $T' = T \cup \{\forall \underline{x} A(\underline{x}, f(\underline{x}))\}$ be a theory in the language generated by $\Sigma' = \Sigma \cup \{f : \underline{a} \to s\}$, where $f : \underline{a} \to s$ is a new, defined function declaration. Then T' is an absolute APC extending T and the inheritance property is satisfied.

Conservative extension b) Under the same hypotheses on T, let $T + \mathrm{IKA} \vdash \forall \underline{x}(B(\underline{x}) \vee \neg B(\underline{x}))$, where B is any formula. Let $T'' = T \cup \{\forall \underline{x}(B(\underline{x}) \leftrightarrow r(\underline{x}))\}$ be any theory in the language generated by $\Sigma'' = \Sigma \cup \{r : \underline{a}\}$, where $r : \underline{a}$ is a new, defined relation declaration. Then T'' is an absolute APC extending T and the inheritance property holds.

Now, we briefly show how our methodology uses the above results. *Expansions by new relations or functions* will be made by statements such as:

Expansion $T_1^{ext}(\ldots)$ of $T(\ldots)$;
Signature:
new rels: $r : \underline{a}$;
new internal axioms: Axr
computability theorems:
proof of $\forall \underline{x}(r(\underline{x}) \vee \neg r(x))$
End Expansion.

where we introduce a new relation declaration $r : \underline{a}$. If we have a conservative extension b), then Axr contains *only* the explicit definition of r. Otherwise, the new axioms Axr must be elementary basic formulas (the old induction schemes are automatically extended, so as to the include all their instances in the new language).

Expansion $T_2^{ext}(\ldots)$ of $T(\ldots)$;
Signature:

new functs: $f : \underline{a} \rightarrow s$;

new internal axioms: $AxRf$;

computability theorems:

proof of $\forall \underline{x} \exists! z Rf(\underline{x}, z)$

End Expansion.

where we introduce a new function declaration $f : \underline{a} \rightarrow s$ and:
i) we make a conservative extension b), $AxRf$ contains only the definition of f and Rf *is the defining formula*;
ii) we make an extension according to Theorem 4.7 and there is also an implicit relation declaration "$Rf : \underline{a}s$" together with an implicit axiom $\forall \underline{x} Rf(\underline{x}, f(\underline{x}))$, and $AxRf$ contains *only* elementary basic formulas.

Function Expansion $T_2^{ext}(\ldots)$ of $T(\ldots)$;

Signature:

new functs: $f : \underline{a} \rightarrow s$;

new internal axioms: Axf;

computability theorems:

f-free proof of $\forall x \exists z f(\underline{x}) = z$ (see explanations below)

End Expansion.

where we do not introduce the implicit relation symbol Rf (i.e. Axf does not contain Rf). Let us explain how Theorem 4.7 can be used in this case.

– Every axiom $A \in Axf$ can be translated into an axiom A_{rel}, containing Rf but not f, such that:
(*) $\{\forall \underline{x}, y : (Rf(\underline{x}, y) \leftrightarrow y = f(\underline{x}))\} + \mathrm{IKA} \vdash A \leftrightarrow A_{rel}$;
we indicate the result of the translation by $AxRf$.

– Let $\Pi(\forall \underline{x} \exists z\ f(\underline{x}) = z)$ be a proof (e.g. in natural calculus) proving in IKA the formula $\forall \underline{x} \exists z\ f(\underline{x}) = z$. We say that Π is *f-free* iff, for every $\exists$-introduction and $\forall$-elimination

$$\frac{H(t)}{\exists x H(x)} \qquad\qquad \frac{\forall x H(x)}{H(t)}$$

used in Π, f *does not occur in the involved term* t. If $\Pi(\forall \underline{x} \exists z\ f(\underline{x}) = z)$ *is* f-free, then, using (*), it can be translated into a proof $\Pi^*(\forall \underline{x} \exists z Rf(\underline{x}, z))$ not containing f and proving $\forall \underline{x} \exists z Rf(\underline{x}, z)$.

REMARK. The restriction to f-free proofs forbids in particular the following trivial proof (which doesn't work as a computability proof of f):

$$\frac{f(x) = f(x) \quad \text{(by identity rule)}}{\exists z f(x) = z}$$

By the above points, if the axioms of the translation $AxRf$ of Axf are elementary basic formulas and the computability proof $\Pi(\forall \underline{x} \exists z f(\underline{x}) = z)$ is f-free, then, by Theorem 4.7, we have a correct expansion. Since f-freeness can be automatically checked and the above translation automatically performed, we obtain a Function Expansion module, where the

computability proofs are often shorter than the associated translations.

Finally, we show the APC-expressions which allow to make instantiations by ADT's (giving rise to ADT's) and by APC's (giving rise to other classes).

If T is an APC with respect to $\mathcal{C}$ and T' completely formalizes an ADT of $\mathcal{C}$, then we have a correct instantiation, expressed by:

Instance $T''(\ldots;) = T(R_1, \ldots, R_n; A_1 :: T', \ldots, A_m :: T')$

where the list of the defined parameters $(\ldots;)$ of T'' is a possibly empty sublist of the ones coming from the instantiation operation (in other words we may eliminate defined parameters). The theory T'' has no input parameters, since all parameters of T have been instantiated. If some input parameter is not instantiated, we cannot use the expression "**Instance** ...", but we have to use the subclass instantiation. The latter is given using the keyword **Subclass** instead of **Instance**.

Instantiations using APC's give rise to APC-composition. APC-composition can be used to get expansions of APC's introducing many functions and relations, as follows.

Let $T(N_1, .., N_k; I_1, .., I_n)$ be an APC such that N_1, .., N_k are function or relation declarations, I_1, .., I_n are relation declarations, and the axioms are either canonical constraints for the input parameters, or formulas defining the relation or function symbols of $N_1, .., N_k$ and satisfying the hypotheses of Theorem 4.7. Let $T^*(D_1, .., D_h; Y_1, .., Y_m)$ be an APC with respect to a class $\mathcal{C}$; then the APC-IKA-composition

Expansion $T^{ext}(D_1, .., D_h, R_1, .., R_k; Y_1, .., Y_m) = T(R_1, .., R_k; A_1 :: T^*, .., A_n :: T^*)$

is an APC with respect to $\mathcal{C}$, expanding T^* (by $R_1, \ldots, R_k$) and satisfying expansion inheritance. If some input parameter is not instantiated, we obtain a pseudo subclass.

4.4. ON THE USE OF PSEUDO LOGICS DIFFERENT FROM IKA

As anticipated above, the previous results can be given in terms of pseudo logics P much more powerful than IKA (even if, passing from $T+$IKA to $T+P$, one may loose the strong constructiveness of $T+P$, or the possibility of extending in a strongly constructive way $T+P$). In this line, in Miglioli *et al.* (1989) three pseudo logics, P_1, P_2 and P_3, having a greater deductive power than IKA are considered:

P_1 (which contains many principles not in IKA among which the well-known Markov's one) gives rise to subconstructive formal systems $T+P_1$ for any $\forall\exists\neg$-theory T *completely formalizing an ADT* (see the 1-systems of Miglioli *et al.* (1989)). P_2 and P_3 give rise to subconstructive formal systems $T+P_2$ and $T+P_3$ for any *atomically complete* $\forall\neg$-theory T (the existence of a reachable model is not necessary); P_2 and P_3 give rise to constructive systems also for atomically complete $\forall\exists - \forall\neg$-theories with a reachable isoinitial model (see, respectively, the 2-systems and the 3-systems of Miglioli *et al.* (1989)). P_1, P_2 and P_3 are mutually semiconstructively incompatible, since there are $\forall\neg$-theories T such that $T + P_1 + P_2 = T + CL$, $T + P_1 + P_3$ and $T + P_2 + P_3$ are not semiconstructive. Now, the above Theorem 4.2 can be stated also for P_1 instead of IKA, while P_2 and P_3 require $\forall\exists - \forall\neg$-theories. The same holds for Theorem 4.3. Theorem 4.4 can be stated, *mutatis mutandis*, using P_1, P_2 or P_3. The same holds for Theorems 4.5, 4.6 and 4.7.

We remark that two other pseudo logics, P_4 and P_5 (both containing Grzegorczyk's

principle) are considered in Miglioli *et al.* (1989), where: P_4 extends both P_2 and P_3; P_5 extends P_1; $T + P_4$ and $T + P_5$ are subconstructive for any *elementary T without non-elementary instances of induction schemes*; P_4 and P_5 are semiconstructively incompatible, i.e., there is an elementary T without non-elementary instances of induction schemes such that $T + P_4 + P_5$ is not semiconstructive. These two pseudo logics (like any pseudo logic containing Grzegorczyk's principle) are quite inadequate for the methodology explained in this paper. For, not only induction is not allowed with them; in general one also has that the addition of canonical constraints to any elementary theory T gives rise to a T' such that $T' + P_4$ and $T' + P_5$ are not semiconstructive.

5. Examples

5.1. CLASSES

Class $Th(S, C1, C2;)$;
Signature:
sorts: S;
consts: $C1, C2 : S$;

internal axioms: S constructed by $C1, C2$;
End Class.

REMARK. "S constructed by C1, C2" gives rise automatically to the constructors internal axioms:

(inj) $\neg C1 = C2$

(indS) $\dfrac{H(C1) \quad H(C2)}{H(x)}$

No constraint is generated and this class is an *improper class*, i.e. it completely formalizes an ADT. The model where the carrier of S contains exactly two elements $C1$, $C2$ is an isoinitial model. We use defined parameters to allow renaming.

Class $Sum(SUM; X, Y)$;
Signature:
sorts: SUM, X, Y;
functs: $J1 : X \rightarrow SUM$;
$J2 : Y \rightarrow SUM$

internal axioms: SUM constructed by $J1, J2$;
constraints, computability theorems: IMPLICIT;

End Class.

REMARK. "SUM constructed by $J1, J2$" gives rise automatically to the axioms:

(inj1) $\forall x, y : \neg J1(x) = J2(y)$
(inj2) $\forall x, y : (J1(x) = J1(y) \rightarrow x = y)$
(inj3) $\forall x, y : (J2(x) = J2(y) \rightarrow x = y)$
(indSUM) $\dfrac{H(J1(x)) \quad H(J2(y))}{H(z)}\, x, y$ †
(constrX) $\forall x, y : X(x = y \vee \neg x = y)$
(constrY) $\forall x, y : Y(x = y \vee \neg x = y)$

The computability proof of $=: SUM \times SUM$ is implicit (we have a pure constructors theory).

Class $Prod(PROD; X, Y)$;
Signature:
sorts: $PROD, X, Y$;
functs: $\langle \ldots, \ldots \rangle : X \times Y \rightarrow SUM$;
internal axioms: $PROD$ constructed by $\langle \ldots, \ldots \rangle$;
constraints, computability theorems: IMPLICIT;
End Class.

REMARK. "PROD constructed by: $\langle \ldots, \ldots \rangle$" gives rise automatically to the axioms:

(inj) $\forall x_1, x_2, y_1, y_2(\langle x_1, y_1 \rangle = \langle x_2, y_2 \rangle \rightarrow (x_1 = x_2 \wedge y_1 = y_2))$
(indPROD) $\dfrac{H(\langle x, y \rangle)}{H(z)}\, x, y$
(constrX) $\forall x, y : X(x = y \vee \neg x = y)$
(constrY) $\forall x, y : Y(x = y \vee \neg x = y)$

The computability proof of $=: PROD \times PROD$ is implicit (we have a pure constructors theory).

5.2. EXPANSIONS

Function Expansion $Prod1(PROD, P1, P2; X, Y)$ **of** $Prod(PROD; X, Y)$;
Signature:
new functs: $P1 : PROD \rightarrow X$;
$P2 : PROD \rightarrow Y$;
new internal axioms:
(def.P1,P2) $\forall x, y : (P1(\langle x, y \rangle) = x \wedge P2(\langle x, y \rangle) = y)$;
computability theorem:

$$\frac{P1(\langle x, y \rangle) = x}{\dfrac{\exists z P1(\langle x, y \rangle) = z}{\exists z P1(u) = z}} \qquad \frac{P2(\langle x, y \rangle) = y}{\dfrac{\exists z P2(\langle x, y \rangle) = z}{\exists z P2(u) = z}}$$

End Expansion.

Here we have a functional expansion. The computability proof of $P1$ is $P1$-free, since

† In (indSUM), x, y indicate the *induction parameters*. As usual, they cannot occur free in the undischarged assumptions.

the term involved in the $\exists$-introduction rule is x, and it does not contain $P1$. The computability proof of $P2$ is $P2$-free (the term involved in the $\exists$-introduction is y). Since no new constraints are required by the computability proofs of this expansion, any (present and future) instantiation of the APC Prod *can inherit its expansion Prod*1.

An expansion introducing new parameters and new constraints, hence giving rise to a pseudo subclass, is the following:

Expansion $Prod2(PROD, \leq: PROD \times PROD;\ X, Y, \leq: X \times X, \leq: Y \times Y)$
of $Prod(PROD; X, Y)$;
Signature:

new rels: $\ldots \leq \ldots : X \times X;$
$\ldots \leq \ldots : Y \times Y;$
$\ldots \leq \ldots : PROD \times PROD;$

new internal axioms:

(def.po1) $\forall x_1, x_2, y_1, y_2((\neg x_2 \leq x_1 \vee (x_1 = x_2 \wedge y_1 \leq y_2)) \rightarrow \langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle)$
(def.po2) $\forall x_1, x_2, y_1, y_2(\neg(\neg x_2 \leq x_1 \vee (x_1 = x_2 \wedge y_1 \leq y_2)) \rightarrow \neg\langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle);$

new constraints:

(decX) $\forall x, y : X(x \leq y \vee \neg x \leq y)$
(decY) $\forall x, y : Y(x \leq y \vee \neg x \leq y)$

computability proofs:

$$\dfrac{\dfrac{\dfrac{\begin{array}{cccc}\neg x_2 \leq x_1^{(t)} & x_1 = x_2^{(t)}\ y_1 \leq y_2^{(t)} & x_2 \leq x_1^{(t)}\ \neg y_1 \leq y_2^{(t)} & x_2 \leq x_1^{(t)}\ \neg x_1 = x_2^{(t)} \\ \vdots & \vdots & \vdots & \vdots \\ \langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle & \langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle & \neg\langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle & \neg\langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle\end{array}}{\langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle \vee \neg\langle x_1, y_1\rangle \leq \langle x_2, y_2\rangle}(test)}{u \leq \langle x_2, y_2\rangle \vee \neg u \leq \langle x_2, y_2\rangle}}{u \leq v \vee \neg u \leq v}$$

End Expansion.

This expansion contains the new overloaded relation $\leq$ and new constraints. It can be inherited only by instances where the new input parameters $\leq: X \times X$, $\leq: Y \times Y$ can be instantiated. In the IKA-proof of the computability formula $u \leq v \vee \neg u \leq v$ of $\leq: PROD \times PROD$, the canonical constraints (decX), (decY) and the canonical constraint on the identity on X are needed, in order to discharge the assumptions marked by the superscript (t) in the computability proof. The applied "test" rule is a derived rule of IKA. It "IKA-discharges the test assumptions" (indicated by (t)), provided that the corresponding canonical constraints are proved (from the axioms of $Prod2$) using IKA, and their disjunction is propositionally valid in CL (in our case, $\neg x_2 \leq x_1 \vee (x_1 = x_2 \wedge y_1 \leq y_2) \vee (x_2 \leq x_1 \wedge \neg y_1 \leq y_2) \vee (x_2 \leq x_1 \wedge \neg x_1 = x_2)$).

5.3. INSTANTIATIONS AND SUBCLASSES

Instance $Persons = Th(PERSONS, John, Mary)$

This instance is sound, since there are no constraints to be verified. It is a renaming of an ADT and does not contain parameters.

Instance $Couples = Prod(COUPLES; PERSONS :: Persons, PERSONS :: Persons)$

The instance Couples *can inherit the expansion* $Prod1$, giving rise to $Couples1$. But it cannot inherit the expansion $Prod2$. This could be partially instantiated, giving rise to

the subclass:

Subclass $Couples2(\leq\!: COUPLES \times COUPLES; \leq\!: PERSONS \times PERSONS) =$
$Prod2($ $COUPLES, \leq\!: COUPLES \times COUPLES; PERSONS :: Persons,$
$PERSONS :: Persons, \leq\!: PERSONS \times PERSONS,$
$\leq\!: PERSONS \times PERSONS$)

Here we have *only* the constraint for the decidability of $\leq$ (the constraints on $=$ have been deleted by the parameter passing, replacing X, Y by $PERSONS$).

Subclass $Prod3(PROD, P1, P2, \leq\!: PROD \times PROD; X, Y, \leq\!: X \times X, \leq\!: Y \times Y) =$
$Prod1 + Prod2$

Prod3 is well defined, since it comes from two expansions of the same class *Prod* which use different new symbols and separate new axioms (+ is defined in the obvious way)

5.4. INSTANCES AND SUBCLASSES DEFINED BY RECURRENCE

Now let us come to the definitions by IKA-recurrence. In order to make the treatment more intuitive, we give some examples before giving, in the next section, the related theory.

In our definitions by recurrence we will use the following class R as a constructor, where both X and S can be replaced by a (possibly new) name N (we will say that X is instantiated *by recurrence on* N):

Class $R(S, n, f; X)$;
Signature:
sorts: S, X;
consts: $n : S$;
functs: $f : X \rightarrow S$;
internal axioms: S constructed by n, f;
constraints, computability theorems: IMPLICIT;
End Class.

S is constructed starting from some null element n and some operation f, starting from X. We can make APC-IKA-compositions combining $R(S, n, f; X)$ with various APC's and then instantiate X by recurrence on S, obtaining different theories formalizing different APC's or ADT's.

We start with the ADT theory formalizing natural numbers:

Recurrence instance $Nat = R(N, 0, succ; N)$.

Here N is both an input and a defined parameter, since X has been instantiated by recurrence on N.

The constraint on $=\!: N \times N$ with respect to the input parameter N is

$$\forall x, y : N(x = y \vee \neg x = y)$$

This constraint coincides with the computability formula of $=\!: N \times N$ with respect to the "output-parameter" N. Hence, it is "proved by recurrence" and disappears.

We remark that the induction rule of $R(S, n, f; X)$

$$\frac{H(n) \quad H(f(x))}{H(z)}\, x$$

(since n becomes 0, f becomes *succ* and the sort X of x becomes the sort S of z) becomes:

$$\frac{\begin{matrix}(H(x))\\ H(0) \quad H(succ(x))\end{matrix}}{H(z)}\, x$$

which is the usual induction rule on the natural numbers.

To obtain lists, we make an APC-IKA-composition of R and $Prod(NELIST; Y, X)$, and then instantiate X by recurrence on $LIST$, as follows:

Recurrence Subclass $List(Y) = R(LIST, nil, J; NELIST :: Prod(NELIST; Y, LIST))$.

Here we have only the input parameter Y and the related constraint. Induction becomes:

$$\frac{\begin{matrix}(H(l))\\ H(nil) \quad H(J(\langle e, l\rangle))\end{matrix}}{H(s)}\, e, l$$

If we take $e|l = J(\langle e, l\rangle)$, we obtain the usual induction on lists.

We remark that lists and natural numbers can be introduced in a more immediate way as constructors theories, as shown in the example of Section 3. For the natural numbers the direct definition is to be preferred. For lists, the inheritance mechanism gives some advantages using the indirect definition. For instance, if in the definition of the theory $List(Y)$ we use the expansion $Prod1$ of $Prod$, we obtain the theory

$List1(Y) = R(LIST, nil, J; NELIST :: Prod1(NELIST, head, tail; Y, LIST))$,

which inherits also the functions

$head \;:\; NELIST \rightharpoonup Y$
$tail \;:\; NELIST \rightharpoonup LIST$

Also, we might try to introduce the instantiation by recurrence of lists using the expansion $Prod2$:

$List2(Y) = R(LIST, nil, J; NELIST :: Prod2(NELIST, \leq: NELIST \times NELIST; Y, LIST, \leq: Y \times Y, \leq: LIST \times LIST))$.

However, in the instantiation by recurrence $\leq: LIST \times LIST$ is an input parameter, not a defined parameter. Thus, we cannot prove by recurrence the constraint on it and the recurrence definition is not sound. To overcome this obstacle, we need the following expansion of R:

Expansion $R1(S, n, J, \leq: S \times S\,;\; X, \leq: X \times X)$ **of** $R(S, n, f; X)$;
signature:
new rels: $\leq: S \times S$;
$\leq: X \times X$;
new internal axioms:

(def1$\leq$ on S) $\forall x, y((x = n \vee (\exists a, b(x = J(a) \wedge y = J(b) \wedge a \leq b)) \rightarrow x \leq y)$
(def2$\leq$ on S) $\forall x, y(\neg(x = n \vee (\exists a, b(x = J(a) \wedge y = J(b) \wedge a \leq b)) \rightarrow \neg x \leq y)$

new constraints:

$\forall x, y : X(x \leq y \vee \neg x \leq y)$

computability proofs:

$\forall x, y : S(x \leq y \vee \neg x \leq y)$ constructively provable from the input constraints
(proof omitted)

End Expansion.

If we assume *non canonical* input constraints stating that $\leq$ is a total ordering on X, then we can prove that $\leq$ is a total ordering on S. Analogously, for $Prod2$ we can prove that, if the input relations $\leq$ are total orderings, then the defined $\leq$ is a total ordering. Notice that, in the instantiation by recurrence

$R1(LIST, nil, J, \leq: LIST \times LIST; NELIST :: Prod2(NELIST, \leq: NELIST \times NELIST; Y, LIST, \leq: Y \times Y, \leq: LIST \times LIST))$

the computability formula of $\leq$ on $LIST$ is proved by recurrence and disappears, whilst the constraint for the input parameter $\leq: Y \times Y$ must be kept. Thus, the instantiation by recurrence $List2(Y)$, using $R1$ and $Prod2$, gives rise to the expansion of $List(Y)$ with the new relation $\leq$. If we add the non canonical constraints stating that $\leq$ is a total ordering, they turn out to be proved by recurrence on $LIST$ and are to be kept only for Y; in this way, we obtain an almost regular class with a total ordering on $LIST$ and $NELIST$.

As a final example, we can provide a formalization of trees by recurrence, using $List(Y)$ and R as follows:

Recurrence instance $Tree(Y) =$
$R(TREE, nilt, tree; NETREE :: Prod(NETREE; Y, TREES :: List(TREE)))$.

Also, using $Prod1(NETREE, root, subtrees; Y, TREES :: List(TREE))$ instead of $Prod$, we obtain an expansion $Tree1(Y)$ of $Tree(Y)$, which inherits the functions:

$$\begin{array}{lcl} root & : & NETREE \rightarrow Y \\ subtrees & : & NETREE \rightarrow TREES \end{array}$$

We remark that, using $R1$ and $Prod2$ in the appropriate way, the theory of trees inherits the relation $\leq$. Using $R1$ and $Prod3$, it inherits $root$, $subtrees$, $\leq$.

The induction rules on $Tree(Y)$ are mutual induction rules (with $t : TREE$, $l, m : TREES$, $y : Y$, $h : NETREE$):

$$\frac{H(nilt) \quad H(tree(h))}{H(t)}\, h \qquad \frac{K(\langle y, l\rangle)}{K(h)}\, y, l \qquad \frac{\begin{array}{c}(P(l)) \\ P(nil) \quad P(t|l)\end{array}}{P(m)}\, t, l$$

6. Sufficient criteria for definitions by recurrence

We start from a theorem related to APC-expressions which can be instantiated by recurrence and give rise to ADT's.

Let $T(D,\ldots;X,R_1,\ldots,R_n)$ be a parametric theory. We say that T is *X/D-inductive* iff:

a) D, X are sort symbols and $R_1,\ldots,R_n$ are input relation declarations;
b) the signature of T does not contain function declarations $f : \underline{a} \to s$ with D occurring in $\underline{a}$, or $g : \underline{b} \to s$ with s different from D and X occurring in $\underline{b}$;
c) T formalizes an APC with respect to some (non-empty) class $\mathcal{C}$;
d) D is *weakly constructed* by a set $\mathcal{K}$ of *basic constants* of sort D, a set $\mathcal{B}$ of *basic function* declarations with sort D and arity not containing the input sort X, and a set $\mathcal{J}$ of *X-function* declarations with sort D and arity containing X, where, by saying that D is weakly constructed by $\mathcal{K}$,$\mathcal{B}$ and $\mathcal{J}$, we mean that:
   - $\mathcal{K}$,$\mathcal{B}$ may be empty, but $\mathcal{B} \cup \mathcal{K} \neq \emptyset$;
   - the induction rule $ind(\mathcal{K},\mathcal{B},\mathcal{J})$ for the sort D, related to $\mathcal{K}$, $\mathcal{B}$, $\mathcal{J}$, is in T;
   - the injectivity axioms may be omitted;
e) the axioms of T are *elementary* formulas.

REMARKS.

(r1) The induction rule $ind(\mathcal{K},\mathcal{B},\mathcal{J})$ is:

$$\frac{\begin{array}{ccc}\text{group1} & \text{group2} & \text{group3}\\ \cdots H(c) \cdots & \cdots H(f(\underline{v})) \cdots & \cdots H(g(\underline{x},\underline{y})) \cdots\end{array}}{H(z)}\ \underline{v},\underline{x},\underline{y}$$

where for every $c \in \mathcal{K}$ we have a subproof of group1, for every $f : \underline{a} \to D \in \mathcal{B}$ we have a subproof of group2 and for every $g : \underline{b} \to D \in \mathcal{J}$ we have a subproof of group3.

The variables of sort X occurring in g are indicated by $\underline{x}$. We say that $\underline{x}$ is *X- involved* in the rule $ind(\mathcal{K},\mathcal{B},\mathcal{J})$.

(r2) Let $T'' = T(D,\ldots;X :: T',A_1 :: T',\ldots,A_n :: T')$ be an instance by $T' \in \mathcal{C}$ (then T'' completely formalizes an ADT I''). If U is a sort different from D and X, then the carrier of U in I'' depends only on T and not on the instance T' (since T'' is reachable and no term of sort U contains X or D in its arity). Analogously, if the arity of a declaration relation $r : \underline{a}$ does not contain D or X, then the diagram $d(r : \underline{a})$ (in I'') is $d(r : \underline{a}) = \{r(\underline{t}) \mid T'' + CL \vdash r(\underline{t})\} \cup \{\neg r(\underline{t}) \mid T'' + CL \vdash \neg r(\underline{t})\} = \{r(\underline{t}) \mid T + CL \vdash r(t)\} \cup \{\neg r(\underline{t}) \mid T + CL \vdash \neg r(t)\}$. We will say that the (possible) sorts of T different from D and X and the relations and functions on them *are defined by* T.

Given a X/D-inductive theory $T(D;X,R_1,\ldots,R_n)$, we define its instantiation by recurrence as the theory T^* (henceforth called the *R-instance* of T) such that:

1) the signature Σ^* of T^* is $\sigma\Sigma$, where $\sigma = \{X/D\}$;
2) the internal axioms $Ax(T^*)$ are $\sigma Ax(T)$, and the new induction rule $ind^*(\mathcal{K},\mathcal{B},\mathcal{J})$ of T^* contains the subproofs of group1 and of group2 of $ind(\mathcal{K},\mathcal{B},\mathcal{J})$, while the subproofs of group3 become:

$$\begin{array}{c}\cdots (H(x_1)) \cdots (H(x_m)) \cdots\\ \cdots\cdots\\ H(f(x_1,\ldots,x_m,\underline{y}))\end{array}$$

where the (possible) *inductive assumptions* $H(x_1),\ldots,H(x_m)$ are discharged by

the rule (we call *induction-steps* the subproofs of group3 and *basic subproofs* the ones of group1 and group2).

Now, we say that a X/D-inductive theory $T(D; X, R_1, ..., R_n)$ *satisfies its constraints by recurrence* in the logic IKA iff its R-instance T^* is such that:

$$T^* + IKA \vdash cc(R)$$

for every relation declaration R.

We have the following theorem.

THEOREM 6.1. *Let $T(D, ...; X, R_1, ..., R_n)$ be a X/D-inductive theory and let T^* be its R-instance. If T satisfies its constraints by recurrence and T^* has a reachable model, then T^* completely formalizes an ADT I^* such that:*

- I^* interprets the (possible) sorts different from D and the (possible) relation- and function-declarations not containing D, according to the above remark (r2);

- the carrier of the sort D is the set of the closed terms of sort D of the signature Σ^*, with a possible quotientation if the injectivity axioms are missing.

Theorem 6.1 follows from remark (r2) and from the fact that $T^* + IKA$ proves the canonical constraints and is constructive. The constructiveness follows from the fact that the axioms of T^* are elementary axioms or instances of the induction schemes.

We remark that points a), b), c), d) play no role in guaranteeing constructiveness. They provide, on the other hand, a clear semantics, where D is the only sort whose carrier is defined by the instantiation by recurrence and the related induction rule is characterized by the weak constructors of D. The definition of a $X/D, S_1, ..., S_m$-inductive theory $T(D, S_1, ..., S_m; X, R_1, ..., R_n)$ can be given, where D, S_1, ..., S_m are simultaneously (possibly weakly) constructed, in such a way that the substitution X/D (corresponding to an instantiation by recurrence) gives rise to a mutual induction schema. In the following, for the sake of simplicity, we will consider only the case of X/D-inductive theories. But what we will say can be generalized to $X/D, S_1, ..., S_m$-inductive theories.

In order that the above theorem can be reasonably used, we need some sufficient conditions which allow to state whether a X/D-inductive theory $T(D; X, R_1, ..., R_n)$ satisfies its constraints by recurrence and whether its R-instance T^* has a reachable model. We have:

Recurrence instantiability 1. Let $T(D; X)$ be the *pure constructors theory* $PT(\Sigma)$ of some signature Σ with input sort X and defined parametric sort D (also other non parametric sorts are allowed). Then T is X/D-inductive, T satisfies its constraints by recurrence and T has a reachable model. Hence, its R-instance T^* completely formalizes an ADT, which is the one formalized by $PT(\Sigma^*)$ (Σ^* is the signature of T^*).

This instantiability property is due to the fact that T^* coincides with $PT(\Sigma^*)$.

Recurrence instantiability 2. Let $T(D; X, R_1, ..., R_n)$ be a *X/D-inductive* theory and let the following requirements be satisfied:

1) T coincides with the *parametric theory* $PT(\Sigma)$ of its signature Σ;

2) for every input relation R_i, its "D-image" σR_i (where $\sigma = \{X/D\}$) belongs to Σ;

3) for every defined relation declaration R, $T + IKA \vdash cc(R)$;

4) every proof Π (in T+IKA) of $cc(\sigma R_i)$ $(1 \leq i \leq n)$ which applies the induction rule $ind(\mathcal{K}, \mathcal{B}, \mathcal{J})$ can be translated into a corresponding proof which applies $ind^*(\mathcal{K}, \mathcal{B}, \mathcal{J})$ in place of $ind(\mathcal{K}, \mathcal{B}, \mathcal{J})$ (the possibility of providing such a translation can be automatically checked and the translation can be automatically generated).

Then $T(D; X, R_1, ..., R_n)$ proves (in IKA) its constraints by recurrence. Moreover, if its R-instance T^* is consistent, then T^* has a reachable model and thus T^* completely formalizes an ADT I^* with the properties of Theorem 6.1 (in particular, since T^* contains the injectivity axioms, the carrier of D is constructed by the closed terms of sort D of the signature of T^*).

The above properties can be proved using the following facts:

-Point 1) implies the reachability of T^*, since T^* becomes $PT(\Sigma^*)$;

-Points 2), 3), 4) imply that $T^* + IKA$ allows to prove by induction the formula $\forall \underline{x}(cc(\sigma R_1) \wedge \cdots \wedge cc(\sigma R_n))$;

- Point 3) implies also the provability of $cc(R)$ for any relation declaration R different from $\sigma R_1, \ldots, \sigma R_n$.

The R-instances constructed according to the above properties could be directly defined as ADT's and the direct definition method is, in the most cases, simpler. Definitions by recurrence are convenient if we are interested in families of ADT's having the same structure. For, in this case inheritance properties are at disposal. As a matter of fact, the inheritance properties of Theorem 4.7 hold also for R-instances, as stated below.

Inheritance properties. Let $T(D, \ldots; X, R_1, ..., R_n)$ be a parametric theory satisfying Recurrence Instantiability 1 or 2. We have:

I) Let $T_1^{ext}(D, r, \ldots; X, R_1, ..., R_n)$ be an extension of T according to Point I) of Theorem 4.7. Then, *mutatis mutandis*, the inheritance property (ip1) of Theorem 4.7 holds also for the ADT I formalized by the R-instance T^* and the one formalized by T_1^{ext*}.

II) Let $T_2^{ext}(D, r, f, \ldots; X, R_1, ..., R_n)$ be an extension of T according to Point II) of Theorem 4.7. Then, *mutatis mutandis*, the inheritance property (ip2) of Theorem 4.7 holds also for the ADT I formalized by the R-instance T^* and the one formalized by T_2^{ext*}.

Other possible advantages of definitions by recurrence depend on the possibility of combining various $X/D, S_1, ..., S_n$-inductive theories in such a way that their recurrence instantiability properties are preserved. We are studying classes of such $X/D, S_1, ..., S_n$-inductive theories. Some examples have been given in the previous section.

References

Bates, J., Constable, R. (1985). Proofs as programs. *ACM Transaction on Programming Languages and Systems* 7, 113–136.

Bertoni, A., Mauri, G., Miglioli, P. (1979). A characterization of abstract data as model-theoretic invariants. In Maurer, H.A. (ed.), *Automata, Languages and Programming*, LNCS 71, pp. 26–37, Springer Verlag.

Bertoni, A., Mauri, G., Miglioli, P. (1980). Towards a theory of abstract data types: a discussion on problems and tools. In Robinet, B. (ed.), *International symposium on programming*, LNCS 83, pp.44–58, Springer-Verlag.

Bertoni, A., Mauri, G., Miglioli, P. (1983). On the power of model theory to specify abstract data types and to capture their recursiveness. *Fundamenta Informaticae* VI(2), 127–170.

Bertoni, A., Mauri, G., Miglioli, P., Ornaghi, M. (1984). Abstract data types and their extension within a constructive logic. In Kahn, G., MacQueen, D.B., Plotkin, G. (eds.), *Semantics of data types*, LNCS 173, pp. 177–195, Springer-Verlag.

Bresciani, P., Miglioli, P., Moscato, U., Ornaghi, M. (1986). PAP: Proofs as Programs (abstract). *Journal of Symbolic Logic* 51, 852–853.

Bundy, A., Smaill, A., Wiggins, G. (1990). The synthesis of logic programs from inductive proofs. In Lloyd, J. (ed.), *Computational logic*, pp. 135–149, Springer-Verlag.

Chang, C.C., Keisler, H.J. (1973). *Model Theory*. Amsterdam: North-Holland.

Constable, R. *et al.* (1986). *Implementing Mathematics with the Nuprl Development System*. N. J., Prentice-Hall.

Degli Antoni, G., Miglioli, P., Ornaghi, M. (1974). Top-down approach to the synthesis of programs. In Robinet, B. (ed.), *Programming Symposium*, LNCS 19, pp. 88–108, Springer-Verlag.

Degli Antoni, G., Miglioli, P., Ornaghi, M. (1975). The synthesis of programs as an approach to the construction of reliable programs. In Kahn, G., Huet, G.(eds.), *Construction, amelioration et verification de programmes. Proving and Improving Programs*, Colloques IRIA, pp. 327–352.

Ehrig, H., Mahr, B., (1985). *Fundamentals of Algebraic Specification 1*. Berlin: Springer-Verlag.

Ehrig, H., Mahr, B., (1989). *Fundamentals of Algebraic Specification 2*. Berlin: Springer-Verlag.

Gabbay, D. (1981). *Semantical investigation in Heyting's intuitionistic logic*. Dordrecht: Reidel.

Goad, C. (1980). *Computational uses of the manipulation of formal proofs*. Stanford University: rep. STAN-CS-80-819.

Goguen, J.A., Thatcher, J.W., Wagner, E.G. (1976). *An initial algebra approach to the specification, correctness and implementation of abstract data types*. Yorktown Heights: IBM Res. Rep. RC6487.

Goguen, J.A. (1984). Parameterized Programming. *Transaction on Software Engineering* SE10(5), 538–543.

Goguen, J.A., Meseguer, J. (1987). Unifying Functional, Object-oriented and Relational Programming with Logical semantics. In Shriver, B., Wegner, P. (eds.), *Research Directions in Object-Oriented Programming*, pp. 417–477, Cambridge: MIT Press.

Goguen, J.A., Burstall, R. (1992). Institution: Abstract Model Theory for specification and programming. *Journal of the Association of Computer Machinery* 39, 95–146.

Goto, S. (1979). Program synthesis from natural deduction proofs. *International Joint Conference on Artificial Intelligence*, pp. 339–341, Tokyo, 1979.

Hayashi, S., Nakano, H. (1988). *PX: A computational logic*. Cambridge: MIT Press.

Henson, M. (1989). Realizability models for program construction. In van de Snepscheut, J. (ed.), *Mathematics of program construction* , LNCS 375, pp. 256–272, Springer-Verlag.

Lloyd, J. (1987). *Foundations of logic programming*. Berlin: Springer-Verlag.

Martin-Löf, P. (1982). Constructive Mathematics and Computer Programming. In Cohen, L., Los, J., Pfeiffer, H., Podewski, K. (eds.), *VI International Congress for Logic, Methodology and Philosophy of Science*, pp.153–175, Amsterdam: North-Holland.

Miglioli, P., Ornaghi, M. (1981). A logically justified model of computation I, II. *Fundamenta Informaticae*, IV(1,2), 1981, 151–172, 277–341.

Miglioli, P., Moscato, U., Ornaghi, M. (1988). Constructive theories with abstract data types for program synthesis. In Skordev, D.G. (ed.), *Mathematical Logic and its Applications*, pp. 293–302, New York: Plenum Press.

Miglioli, P., Moscato, U., Ornaghi, M. (1988 bis). PAP: a logic programming system based on a constructive logic. In Boscarol, M., Carlucci Aiello, L., Levi, G. (eds.), *Foundations of logic and functional programming*, LNCS 306, pp. 143–156, Springer-Verlag.

Miglioli, P., Moscato, U., Ornaghi, M. (1989). Semi-constructive formal systems and axiomatization of abstract data types. In Diaz, J., Orejas, F. (eds.), *TAPSOFT '89*, LNCS 351, pp. 337–351, Springer Verlag.

Miglioli, P., Moscato, U., Ornaghi, M. (1992). Program specification and synthesis in constructive formal systems. In Lau, K.-K., Clement, T.P. (eds.), *Logic Program Synthesis and Transformation, Manchester 1991*, pp. 13–26, Workshops in Computing, Springer Verlag.

Prawitz, D. (1965). *Natural deduction: a proof-theoretical study*. Stockholm: Almqvist & Wiksell.

Sato, M. (1979). Towards a mathematical theory of program synthesis. *International Joint Conference on Artificial Intelligence*, Tokyo, pp. 757–762.

Troelstra, A.S. (ed.) (1973). *Metamathematical investigation of Intuitionistic Arithmetic and Analysis*. LNM 344, Berlin: Springer-Verlag.

Voronkov, A. (1990). Toward the theory of programming in constructive logic. In Jones, N. (ed.), *ESOP '90*, LNCS 432, pp. 421–435, Springer Verlag.

Wirsing, M. (1990). Algebraic specification. In Van Leeuwen, J. (ed.), *Handbook of Theoretical Computer Science*, pp. 675–788, Amsterdam: Elsevier.