On constructive specifications of abstract data types using temporal logic

Frank Leßke *
Institut für Informatik
Ludwig Maximilians Universität, Theresienstr.39, 8000 München

Oktober 1991

Abstract

We introduce an approach for the specification of abstract data types based on temporal logic. To this end we propose a constructive specification method. We present axiom schemes to get generally monomorphic and complete models. Non-constructive operations are defined as abbrevations for algorithms using constructors. We show that our approach is as expressive as the classical method. Moreover we can specify semicomputable and co-semicomputable algebras monomorphically.

1 Introduction

A well known problem in the area of program development is the specification and analysis of abstract data types. On the other hand, in the last years temporal logic emerged to be a promising formalism for the description of the behaviors of state based systems. In this paper we want to show how temporal logic formulae may be used for the specification of data types.
In the classical viewpoint a data type is a multisorted algebra consisting of a set of carrier sets and a set of operations on these carrier sets. One can distinguish two different methods for the specification of abstract data types. In the algebraic approach a data type is regarded as a specific set of models, as e.g the initial ones, for a set of formulae of first-order logic or of a subset of it like equational logic. In the operational or constructive method the carrier sets are constructed in an imperative programming language from the therein available data, defining the operations as procedures. Algebraic specifications are more abstract, so one can avoid overspecifications and verification gets easier. But they lead to several problems concerning consistency and sufficient completeness of the

*The work of the author is supported by the Sonderforschungsbereich 342 of the TU München.

axiom sets, enrichment and extension of the data types, and the treatment of partial functions. To avoid these problems there exist different approaches to combine the algebraic framework with an applicative language in order to get abstract and algorithmic specifications, like e.g. in [Klae84],[Loe87] or [Mir87].
We want to follow mainly the work of [Min88] who has used second-order logic for constructive specifications. Instead we will use the weaker temporal logic. We use it as an algorithmic language with the usual abstract semantics. We show that in our framework we can state general axiom schemes for a unique description of any data type. In this sense we generalize the work of [Sza88], who has only reached monomorphic specifications for specific data types as e.g stacks. Similiar results as ours appear in [Merz91] who uses a different temporal logic and a different specification method to characterize initial models. We also will show that with our axiom sets the specifications are complete with respect to arbitrary temporal logic formulae.
If one deals with constructive specifications one comes up with the problem of enrichment, that means the definition of a new operation as abbrevation for a fixed algorithm in terms of the existing operations. We will work out a method for specifying different data types which are based on the same set of objects, but contain different sets of operations. [Krö90] has already shown that unlike in first-order logic in temporal logic it is possible to define e.g. multiplication in Presburger arithmetic. We will show that based on the constructors *zero* and *successor* it is possible to define any recursive function in our framework. We also will see that our approach fits well for the definition of partial functions.
As not every data type can be described as the term algebra of a set of constructors we will then extend our specification method to capture also such types like e.g. SET. We will show then that we can specify monomorphically any data type in our framework which may be specified by a finite set of equations in the initial approach. This means that focussing only on constructive operations is not really a restriction.
On the other hand we will prove that when using temporal logic the expressive power of data specifications increases. Differently from the classical case we can state monomorphic and complete specifications for semicomputable and co-semicomputable algebras.

2 Specification by Constructive Extension of a Signature

2.1 A Temporal Language for Specification

First let us recall some notions about syntax and semantics.
A *signature* $\mathbf{\Sigma} = (\mathbf{S}, \mathbf{F})$ is given by a set of sorts $\mathbf{S} = \{s_1, ..., s_k\}$, for easier notation $\mathbf{S} = \{1, ..., k\}$, and a set $\mathbf{F}$ of function and predicate symbols. The arity of functions and predicates is denoted by $< i_1, ..., i_n >$, $1 \leq i_l \leq k$ and $n \geq 1$. A constant symbol of sort

i is a function symbol with arity $< i >$. We assume $\mathbf{F}$ to be sorted, $\mathbf{F} = \mathbf{F}^1 \cup ... \cup \mathbf{F}^k$, where $\mathbf{F}^l$ is the set of symbols with arity $< i_1, ..., i_n >$, where at least one $i_m = l$ and $i_j \leq l$ for all $1 \leq j \leq n$.
A *signature morphism* $\sigma : \mathbf{\Sigma} \rightarrow \mathbf{\Sigma}'$ is a pair $< \sigma_S, \sigma_F >$ where $\sigma_S : \mathbf{S} \rightarrow \mathbf{S}'$ and $\sigma_F : \mathbf{F} \rightarrow \mathbf{F}'$ are mappings, such that for any $f \in \mathbf{F}$ with arity $< i_1, ..., i_n >$ the arity of $\sigma_F(f)$ is $< \sigma_S(i_1), ..., \sigma_S(i_n) >$.

Let X be a given $\mathbf{S}$-sorted set of variables. A *term* of sort i is a constant symbol of sort i or a variable of sort i or a function symbol with arity $< i_1, ..., i_{n-1}, i >$ applied to arguments which are terms of the respective sorts. The set of all terms of sort i is denoted by $\mathbf{T_{\Sigma,i}}(X)$ and the set of all terms by $\mathbf{T_\Sigma}(X)$. Terms without variables are called *ground terms* and the respective sets are denoted by $\mathbf{T_{\Sigma,i}}$ and $\mathbf{T_\Sigma}$.

For a signature $\mathbf{\Sigma} = (\mathbf{S}, \mathbf{F})$ a *temporal logic language* $\mathbf{TL_\Sigma}$ is defined. The alphabet of $\mathbf{TL_\Sigma}$ is $\mathbf{F}$ with additional denumerable $\mathbf{S}$-sorted sets of global and local variables [1], the (binary) equality symbol $=$, the boolean and temporal connectives (e.g. $\neg$, $\wedge$, $\bigcirc$, $\Box$), quantifiers and brackets. Atomic formulae are defined as usual by application of predicate symbols or the symbol $=$ to terms of admissible sorts. Formulae are formed by applying the logical and temporal connectives and additional by the following two quantification rules :

($\forall_1$) if A is a formula and x^i is a global variable of sort i then $\forall x^i A$ is a formula

($\forall_2$) if A is a formula and a^i is a local variable of sort i then $\forall a^i A$ is a formula.

For more details about the construction of formulae see e.g. [Krö87a].

For a signature $\mathbf{\Sigma} = (\mathbf{S}, \mathbf{F})$ a Σ-*algebra* $\mathbf{A} = (|\mathbf{A}|, \mathbf{F^A})$ consists of a $\mathbf{S}$-sorted family of non empty carrier sets for each sort, $|\mathbf{A}| = \{|\mathbf{A}|^i\}_{i \in \mathbf{S}}$, and a set $\mathbf{F^A}$ of operations and relations, with a total operation $f^{\mathbf{A}} : |\mathbf{A}|^{i_1} \times ... \times |\mathbf{A}|^{i_{n-1}} \rightarrow |\mathbf{A}|^{i_n}$ for each function symbol $f \in \mathbf{F}$ with arity $< i_1, ...i_{n-1}, i_n >$ and a relation $p^{\mathbf{A}} \subseteq |\mathbf{A}|^{i_1} \times ... \times |\mathbf{A}|^{i_n}$ for each predicate symbol $p \in \mathbf{F}$ with arity $< i_1, ..., i_n >$.
The *term algebra* $\mathbf{T_\Sigma}(X)$ [2] with respect to an $\mathbf{S}$-sorted set X of variables has carrier sets $|\mathbf{T_\Sigma}(X)|^i = \mathbf{T_{\Sigma,i}}(X)$ and operations $f^{\mathbf{T_\Sigma}(X)} : \mathbf{T_{\Sigma,i_1}}(X) \times ... \times \mathbf{T_{\Sigma,i_{n-1}}}(X) \rightarrow \mathbf{T_{\Sigma,i_n}}(X)$ defined by $f^{\mathbf{T_\Sigma}(X)}(t_1, ..., t_{n-1}) := f(t_1, ..., t_{n-1})$ for $f \in \mathbf{F}$ with arity $< i_1, ..., i_n >$ and $t_r \in \mathbf{T_{\Sigma,i_r}}(X)$, $1 \leq r \leq n-1$. The algebra of ground terms $\mathbf{T_\Sigma}(\emptyset)$ is then also denoted by $\mathbf{T_\Sigma}$.
For a $\mathbf{\Sigma}'$-algebra $\mathbf{A}'$ and a signature morphism $\sigma : \mathbf{\Sigma} \rightarrow \mathbf{\Sigma}'$ the σ-*reduct* of $\mathbf{A}'$, written $\mathbf{A}'|_\sigma$, is the $\mathbf{\Sigma}$-algebra with carrier set $|\mathbf{A}'|_\sigma|^i = |\mathbf{A}'|^{\sigma(i)}$ for each sort $i \in \mathbf{S}$ and $f^{\mathbf{A}'|_\sigma} = \sigma(f)^{\mathbf{A}'}$ for each $f \in \mathbf{F}$.

[1] The sorts of the variables may usually be determined from the context or, if not, they will be characterized by an upper index.

[2] For simplicity we use here again the same denotation as for the set of terms, defined above.

If $\mathbf{\Sigma} \subseteq \mathbf{\Sigma}'$ and σ is the canonical injection (i.e. $\sigma(x) = x$ for all $x \in \mathbf{S} \cup \mathbf{F}$) we call $\mathbf{A}'|_\sigma$ the $\mathbf{\Sigma}$-*restriction* of $\mathbf{A}'$ and denote it by $\mathbf{A}'|_{\mathbf{\Sigma}}$.

A *temporal structure* $\mathbf{K} = (\mathbf{A}, \xi, \mathbf{W})$ for the language $\mathbf{TL}_{\mathbf{\Sigma}}$ is given by a $\mathbf{\Sigma}$-algebra $\mathbf{A}$, a valuation ξ of global variables and a denumerable sequence $\mathbf{W} = \{\eta_0\eta_1\eta_2...\}$ of valuations of local variables with respect to $|\mathbf{A}|$. For any $i \in I\!N_0$ a value $\mathbf{K}_i(t) \in |\mathbf{A}|$ for every term t and a truth value $\mathbf{K}_i(A) \in \{\mathbf{f}, \mathbf{t}\}$ for every formula A is defined as usual (compare e.g. [Krö90]):

$\mathbf{K}_i(x) = \xi(x)$ for any global variable x
$\mathbf{K}_i(a) = \eta_i(a)$ for any local variable a
$\mathbf{K}_i(f(t_1, ..., t_n)) = f^{\mathbf{A}}(\mathbf{K}_i(t_1), ..., \mathbf{K}_i(t_n))$
$\mathbf{K}_i(p(t_1, ..., t_n)) = \mathbf{t} \Leftrightarrow (\mathbf{K}_i(t_1), ..., \mathbf{K}_i(t_n)) \in p^{\mathbf{A}}$
$\mathbf{K}_i(t_1 = t_2) = \mathbf{t} \Leftrightarrow \mathbf{K}_i(t_1) = \mathbf{K}_i(t_2)$, where t_1, t_2 are terms of the same sort
$\mathbf{K}_i(\neg A) = \mathbf{t} \Leftrightarrow \mathbf{K}_i(A) = \mathbf{f}$
$\mathbf{K}_i(A \wedge B) = \mathbf{t} \Leftrightarrow \mathbf{K}_i(A) = \mathbf{t}$ and $\mathbf{K}_i(B) = \mathbf{t}$
$\mathbf{K}_i(\circ A) = \mathbf{t} \Leftrightarrow \mathbf{K}_{i+1}(A) = \mathbf{t}$
$\mathbf{K}_i(\Box A) = \mathbf{t} \Leftrightarrow \mathbf{K}_j(A) = \mathbf{t}$ for all $j \geq i$
$\mathbf{K}_i(\Diamond A) = \mathbf{t} \Leftrightarrow \mathbf{K}_j(A) = \mathbf{t}$ for some $j \geq i$
$\mathbf{K}_i(\forall x A) = \mathbf{t} \Leftrightarrow \mathbf{K'}_i(A) = \mathbf{t}$ for every $\mathbf{K}' = (\mathbf{A}, \xi', \mathbf{W})$ with $\xi \approx_x \xi'$
$\mathbf{K}_i(\forall a A) = \mathbf{t} \Leftrightarrow \mathbf{K'}_i(A) = \mathbf{t}$ for every $\mathbf{K}' = (\mathbf{A}, \xi, \mathbf{W}')$ with $\mathbf{W} \approx_a \mathbf{W}'$,

where the relations $\approx_x$ and $\approx_a$ are defined as follows :

$\xi \approx_x \xi' \Leftrightarrow \xi(z) = \xi'(z)$ for every z other than x
$\mathbf{W} \approx_a \mathbf{W}' \Leftrightarrow \eta_j(b) = \eta'_j(b)$ for every $j \geq 0$ and b other than a.

Note that $\Diamond A$ is equivalent to $\neg\Box\neg A$. We also will use additional connectives and quantifiers as e.g. $A \rightarrow B$ or $\exists x A$ like in classical logic.
A formula A is called *valid* in $\mathbf{K}$, or $\mathbf{K}$ *satisfies* A, or $\mathbf{K}$ is a *model* for A (written as $\mathbf{K} \models A$), if $\mathbf{K}_0(A) = \mathbf{t}$. $\mathbf{K}$ is a model for a set F of formulae (or $\mathbf{K}$ satisfies F) iff it is a model for every single formula in F. A formula B is a *semantic consequence* of a set F of closed formulae iff every model for F is also a model for B (written as $F \models B$). We call a $\mathbf{\Sigma}$-algebra $\mathbf{A}$ a *data model* of a formula B, written $\mathbf{A} \models B$, if there is a $\mathbf{W}$ such that every temporal structure $\mathbf{K} = (\mathbf{A}, \xi, \mathbf{W})$ with arbitrary ξ satisfies B.

2.2 Direct Extensions

A *data specification* $\mathbf{SP} = (\mathbf{\Sigma}, E)$ is given by a signature $\mathbf{\Sigma}$ and a set E of formulae of $\mathbf{TL}_{\mathbf{\Sigma}}$. We assume E to be sorted $E = E^1 \cup ... \cup E^k$, where E^l is the set of formulae which contains symbols from $\mathbf{F}^l$ and possibly symbols from $\mathbf{F}^j$ for $j < l$, but no symbols from $\mathbf{F}^j$ for $j > l$.

Definition 1 *A $\mathbf{\Sigma}$-algebra $\mathbf{A}$ is a* data model *of a specification $\mathbf{SP} = (\mathbf{\Sigma}, E)$, if there is $\mathbf{W}$ such that for all ξ, $\mathbf{K} = (\mathbf{A}, \xi, \mathbf{W})$ satisfies E.*

For a given data model $\mathbf{A}$ of a specification $\mathbf{SP}$ we call $\mathbf{W}$ a *valid state sequence* if for all ξ the structure $\mathbf{K} = (\mathbf{A}, \xi, \mathbf{W})$ satisfies E.

For two $\mathbf{\Sigma}$-algebras $\mathbf{A}$ and $\mathbf{A'}$ a $\mathbf{\Sigma}$-*homomorphism* $\varphi : |\mathbf{A}| \rightarrow |\mathbf{A'}|$ is a family of mappings $\{\varphi^i : |\mathbf{A}|^{\mathbf{i}} \rightarrow |\mathbf{A'}|^{\mathbf{i}}\}_{i \in \mathbf{S}}$, such that for all $t_r \in |\mathbf{A}|^{\mathbf{r}}$, $1 \leq r \leq n$, and for all function symbols $f \in \mathbf{F}$ with arity $< i_1, ..., i_n, i >$:

$$\varphi^i(f^{\mathbf{A}}(t_1, ...t_n)) = f^{\mathbf{A'}}(\varphi^{i_1}(t_1), ..., \varphi^{i_n}(t_n))$$

and for all predicate symbols $p_j \in \mathbf{F}$:

$$(t_1, ..., t_n) \in p_j^{\mathbf{A}} \Leftrightarrow (\varphi^{i_1}(t_1), ..., \varphi^{i_n}(t_n)) \in p_j^{\mathbf{A'}}.$$

A homomorphism φ is called isomorphism if it is bijective.

Lemma 1 *Let $F(x_1, ..., x_n, a_1, ..., a_m)$ be a formula with free global variables x_i and free local variables a_i and φ an isomorphism between two Σ-algebras $\mathbf{A}$ and $\mathbf{A'}$. Then :*

$\mathbf{K} \models F(x_1, ..., x_n, a_1, ..., a_m)$ *for a structure* $\mathbf{K} = (\mathbf{A}, \xi, \mathbf{W})$ $\Leftrightarrow$ $\mathbf{K'} \models F(x_1, ..., x_n, a_1, ..., a_m)$ *for a structure* $\mathbf{K'} = (\mathbf{A'}, \xi', \mathbf{W'})$ *with* $\xi'(x_i) = \varphi(\xi(x_i))$ *and* $\eta'_j(a_l) = \varphi(\eta_j(a_l))$.

Proof : Let $\mathbf{K}$ and $\mathbf{K'}$ be two temporal structures with $\xi'(x_i) = \varphi(\xi(x_i))$ and $\eta'_j(a_l) = \varphi(\eta_j(a_l))$. One can easily show that for every term t : $\varphi(\mathbf{K}_i(t)) = \mathbf{K'}_i(t)$. Then by structural induction over the formulae one may prove that $\mathbf{K}_i(G(x_1, ..., x_n, a_1, ..., a_m)) = \mathbf{t}$ iff $\mathbf{K'}_i(G(x_1, ..., x_n, a_1, ..., a_m)) = \mathbf{t}$ for any formula G with free global variables $x_1, ..., x_n$ and free local variables $a_1, ..., a_m$. This holds especially for $F(x_1, ..., x_n, a_1, ..., a_m)$ in the first state and thus the assertion follows. ♣

Definition 2 *A data specification* $\mathbf{SP}$ *is called* monomorphic, *if all data models of* $\mathbf{SP}$ *are isomorphic.*

We construct the data specifications by extension of the signature and by adding further axioms.

Definition 3 *Let* $\mathbf{SP_1} = (\mathbf{\Sigma_1}, E_1)$ *and* $\mathbf{SP_2} = (\mathbf{\Sigma_2}, E_2)$ *be two data specifications. If their sorts are* $\{1, ..., k\}$ *and* $\{1, .., k, k+1\}$, *respectively,* $F_1^i = F_2^i$ *and* $E_1^i = E_2^i$ *for* $1 \leq i \leq k$, *then* $\mathbf{SP_2}$ *is called* direct extension *of* $\mathbf{SP_1}$.

If $\mathbf{SP}$ is one-sorted then it may be regarded as a direct extension of the empty specification which is monomorphic, as its only model consists of the empty set for the sorts and for the operations, respectively.

Definition 4 *Let* $\mathbf{SP_2}$ *be a direct extension of* $\mathbf{SP_1}$. *Let* $\mathbf{A_1}$ *and* $\mathbf{A_2}$ *be a data model of* $\mathbf{SP_1}$ *and* $\mathbf{SP_2}$. $\mathbf{A_2}$ *is called* direct extension *of* $\mathbf{A_1}$ *with respect to* $\mathbf{SP_2}$, *if* $|\mathbf{A_1}|^i = |\mathbf{A_2}|^i$ *and* $\mathbf{F_{A_1}}^i = \mathbf{F_{A_2}}^i$ *for all* $1 \leq i \leq k$.

We can show that it is enough to consider only the direct extensions of the data models of an extended specification.

Theorem 1 *Let* $\mathbf{SP_2}$ *be a direct extension of a monomorphic data specification* $\mathbf{SP_1}$ *and* $\mathbf{A_1}$ *a data model of* $\mathbf{SP_1}$. *If all direct extensions of* $\mathbf{A_1}$ *with respect to* $\mathbf{SP_2}$ *are isomorphic, then* $\mathbf{SP_2}$ *is monomorphic.*

Proof : One can easily adapt the proof for the second-order logic in [Min88] to our framework.
Let $\mathbf{A_2} = \{\{|\mathbf{A}|^{\mathbf{1}}, ..., |\mathbf{A}|^{\mathbf{k+1}}\}, \{\mathbf{F}^i, 1 \leq i \leq k+1\}\}$ be any data model for $\mathbf{SP_2}$. Then $\{\{|\mathbf{A}|^{\mathbf{1}}, ..., |\mathbf{A}|^{\mathbf{k}}\}, \{\mathbf{F}^i, 1 \leq i \leq k\}\}$ is a model for $\mathbf{SP_1}$ and is isomorphic to $\mathbf{A_1}$. As $\mathbf{A_2}$ is a direct extension, it has to be isomorphic to all direct extensions of $\mathbf{A_1}$ and thus the theorem follows. ♣

2.3 Monomorphic and Complete Specifications

At next we want to consider how direct extensions can be made monomorphic in a schematic manner. Therefore the constructive approach for the specification of abstract data types is chosen. We now will treat only data types for which the intended models are isomorphic to the term algebra of a set of function symbols, the so called free constructors. Later on we will show how to specify monomorphically the initial model of any data type which may be defined by equational specification.
In the following let $\mathbf{SP_1}$ be a monomorphic specification with sorts $\{1, ..., k\}$. Let $\mathbf{SP_2}$ be a direct extension by a sort $k+1$ and a set of constructor symbols which consists of constants $c_1, ..., c_l$ $(l > 0)$ of sort $k+1$ and functions $f_1, ..., f_m$ where each f_j has arity $< i_1, ..., i_{v_j}, ..., i_{n_j}, k+1 >$ with $1 \leq i_1, ..., i_{v_j} \leq k$ and $i_{v_j+1} = ... = i_{n_j} = k+1$ and $n_j \geq 1$.
We will state three kinds of axioms or axiom schemes for the constructive definition of the objects of the intended data models. First we want to restrict our models to be term-generated with respect to the new sort. Therefore let x be a global and a be a local variable of sort $k+1$.

$$\begin{aligned}(\text{CONS})\ \forall x \exists a (a = x \wedge \Diamond\Box \textstyle\bigvee_{i=1}^{l} a = c_i \wedge \\ \Box(\textstyle\bigvee_{i=1}^{l} a = c_i \vee \\ \textstyle\bigvee_{j=1}^{m} \exists y_1 ... \exists y_{v_j} \exists x_{v_j+1} ... \exists x_{n_j} \ ((\textstyle\bigwedge_{r=v_j+1}^{n_j} \bigcirc\Diamond a = x_r) \wedge \\ a = f_j(y_1 ... y_{v_j}, x_{v_j+1} ... x_{n_j}))))\end{aligned}$$

where $y_1 ... y_{v_j}$ are global variables of sort $\leq k$ and $x_{v_j+1} ... x_{n_j}$ are global variables of sort $k+1$.

The next two axiom sets assure that two ground terms of sort $k+1$ are only interpreted by the same objects in a model if they are syntactically equal (We use $\forall(*)$ as abbrevation for the universal closure over all free global variables) :

(DIS) $\forall(*)\ (f_i(x_{i_1},...,x_{i_n}) \neq f_j(y_{j_i},...,y_{j_n}))$
where f_i and f_j are two different constructors including constants,

(ID) $\forall(*)(f_j(x_1,...,x_n) = f_j(y_1,...,y_n) \leftrightarrow x_1 = y_1 \wedge ... \wedge x_n = y_n)$
for every constructor function f_j.

Theorem 2 *Let* $\mathbf{SP_2} = (\mathbf{\Sigma_2}, E_2)$ *be a direct extension with constructors of a monomorphic specification* $\mathbf{SP_1}$ *with new sort* $k+1$. *If* E_2^{k+1} *is the set of the respective instances of (CONS),(DIS) and (ID) then* $\mathbf{SP_2}$ *is monomorphic.*

Proof : Let $\mathbf{A_1} = (|\mathbf{A_1}|, \mathbf{F}_1)$ be a data model of $\mathbf{SP_1}$. From the assumptions all other data models of $\mathbf{SP_1}$ are isomorphic.
We define the set $|\mathbf{A_2}|^{k+1}$ inductively :

1. $c_i \in |\mathbf{A_2}|^{k+1}$ for $1 \leq i \leq l$
2. for each function symbol f_j, $1 \leq j \leq m$:
   if $t_r \in |\mathbf{A_1}|^r$, $1 \leq r \leq v_j$
   and $t_r \in |\mathbf{A_2}|^{k+1}$, $v_j + 1 \leq r \leq n_j$
   then $f_j(t_1,...,t_{n_j}) \in |\mathbf{A_2}|^{k+1}$

Herewith we can construct an algebra $\mathbf{A_2} = (|\mathbf{A_1}| \cup |\mathbf{A_2}|^{k+1}, \mathbf{F}_1 \cup \{c_1,...,c_l, f_1,...,f_m\})$ where the operations of the extended algebra are defined as usual on the term algebra. Then $\mathbf{A_2}$ is a data model of $\mathbf{SP_2}$.
We will show : every direct extension of $\mathbf{A_1}$ with respect to $\mathbf{SP_2}$ is isomorphic to $\mathbf{A_2}$. Because of Theorem 1 it is sufficient to consider only this type of models.
Let $\mathbf{A_2'} = (|\mathbf{A_1}| \cup |\mathbf{A_2'}|^{k+1}, \mathbf{F}_1 \cup \{c_1^{\mathbf{A_2'}},...,c_l^{\mathbf{A_2'}}, f_1^{\mathbf{A_2'}},...,f_m^{\mathbf{A_2'}}\}$ be a data model of $\mathbf{SP_2}$ and direct extension of $\mathbf{A_1}$.
We define a mapping $\varphi = (\varphi^1,...,\varphi^k,\varphi^{k+1})$ from $\mathbf{A_2}$ to $\mathbf{A_2'}$. For $i = 1...k$, φ^i is the identity on $|\mathbf{A_1}|^i$, for the new sort $k+1$, $\varphi^{k+1}(c_t) = c_t^{\mathbf{A_2'}}$ and $\varphi^{k+1}(f_j(t_1,...,t_{n_j})) = f_j^{\mathbf{A_2'}}(\varphi^{i_1}(t_1),...,\varphi^{i_{n_j}}(t_{n_j}))$. Obviously φ is a homomorphism. Since $\mathbf{A_2'}$ satisfies the (CONS) axiom it follows that φ^{k+1} is surjective. Thus from the validity of the (DIS) and (ID) axioms in $\mathbf{A_2'}$ it follows that φ^{k+1} is bijective and therefore φ is an isomorphism. ♣

If we compare our axioms with the work of [Min88], we find that we differ in the (CONS)-axiom. While [Min88] uses a general induction principle, formulated in second-order logic, we only describe the construction of valid terms for which our temporal logic suffices. Of course, as our (CONS) axiom restricts the set of models to reachable (or term-generated) algebras, it should be possible to derive a term induction rule in an adequate formal system for our logic (e.g. the one from [Krö87a] extended with respect to quantification over local variables). A derivation of such a term induction rule for a different temporal framework is given in [Merz91] who uses a temporal logic with

flexible predicates. Compared to us his axioms are declarative, since he collects the whole intended set of objects at once, while our formulae define an algorithm for how to construct each single object. Differently from our constructive approach [Merz91] does not propose a specification method, but shows how to characterize initial models, where the temporal logic plays the role of a meta language and not of a specification language.

Interesting properties of a specification are consistency and completeness. In our work the notion of consistency is the classical one and can be verified by checking the existence of a model. To deal with completeness we first want to define our understanding of a complete specification.

Definition 5 *Let* $\mathbf{SP} = (\mathbf{\Sigma}, E)$ *be a data specification. We say that* $\mathbf{SP}$ *is* complete *iff for every closed formula* A *from* $\mathbf{TL_\Sigma}$ *either* $E \models A$ *or* $E \models \neg A$.

Theorem 3 *Let* $\mathbf{SP_2}$ *be a direct extension of a monomorphic* $\mathbf{SP_1}$ *with new sort* $k+1$. *If* E_2^{k+1} *consists of the respective instances of (CONS),(DIS) and (ID), and there is a data model for* $\mathbf{SP_2}$, *then* $\mathbf{SP_2}$ *is complete.*

Proof : In our proof we will follow the ideas from [Sza88] and extend them to our framework.
First we will prove that for any $\mathbf{\Sigma_2}$-algebra $\mathbf{A}$ which is a data model for $\mathbf{SP_2}$ it follows that $\mathbf{A}$ is a data model of a closed formula B from $\mathbf{TL_{\Sigma_2}}$ iff $E_2 \models B$.
"$\Rightarrow$" : From the assumptions $\mathbf{A}$ is a data model for E_2 and B. If it is not the case that $E_2 \models B$ then there exists a $\mathbf{\Sigma_2}$-algebra $\mathbf{A'}$ and a temporal structure $\mathbf{K'} = (\mathbf{A'}, \xi', \mathbf{W'})$ such that $\mathbf{K'} \models E_2$ and $\mathbf{K'} \models \neg B$. Thus $\mathbf{A'}$ is a data model for $\mathbf{SP_2}$ and because of Theorem 2 isomorphic to $\mathbf{A}$. Then from $\mathbf{K'} \models \neg B$ we get that there exists a state sequence $\mathbf{W}^*$ and a global variable valuation ξ^* such that $\mathbf{K}^* \models \neg B$ for $\mathbf{K}^* = (\mathbf{A}, \xi^*, \mathbf{W}^*)$. As B is a closed formula this is in contradiction with the assumption $\mathbf{A} \models B$.
"$\Leftarrow$" : Since $\mathbf{A} \models E_2$ there has to be a state sequence $\mathbf{W}$ such that $\mathbf{K} \models E_2$ for $\mathbf{K} = (\mathbf{A}, \xi, \mathbf{W})$ for all ξ. Suppose $E_2 \models B$ and $\mathbf{A}$ is not a data model of B. This means there exists a ξ' such that $\mathbf{K'} \models \neg B$ for $\mathbf{K'} = (\mathbf{A}, \xi', \mathbf{W})$. Thus $\mathbf{K'} \models E_2$ and $\mathbf{K'} \not\models B$ which means $E_2 \not\models B$ and contradiction is reached.

Now we are ready to prove that $\mathbf{SP_2}$ is complete. Let $\mathbf{A}$ be a data model which exists from the assumptions. Then from the above it follows for any closed formula B from $\mathbf{TL_{\Sigma_2}}$ that $\mathbf{A}$ is a data model of B iff $E_2 \models B$. Suppose that neither $E_2 \models B$ nor $E_2 \models \neg B$. Thus $\mathbf{A}$ is not a data model of B and not of $\neg B$, which cannot be true since B is universally closed with respect to global and local variables. ♣

One should note that in first-order logic with induction it is not possible to obtain first-order complete axiomatisations of data types. We here have generalised the work of [Sza88] who has reached first-order completeness only for some specific specifications, while we have shown how to state complete axiom sets for any data type. Since we also have quantification over local variables our specifications are complete not only for

classical first-order formulae, but with respect to arbitrary universally closed temporal logic formulae.

So now we are able to build monomorphic and complete specifications starting from the empty specification.
To specify a data type **SP** we will use the following notation :

abstract data type SP **is**
extension of :
basic sorts :
new sort :
constructors :
axioms :

Example 1 A specification NAT of the natural numbers is given as a direct extension of the empty specification :

abstract data type NAT **is**
new sort : $\underline{nat}$
constructors : $0 : \longrightarrow \underline{nat}$
$s : \underline{nat} \longrightarrow \underline{nat}$
axioms : $\forall x \exists a(a = x \wedge \Diamond\Box a = 0 \wedge \Box(a = 0 \vee \exists z(\bigcirc\Diamond\ a = z \wedge a = s(z))))$
$\forall x\ 0 \neq s(x)$
$\forall x \forall y\ s(x) = s(y) \leftrightarrow x = y.$

If we take $\mathbf{N}$ as the data model with $|\mathbf{N}| = I\!N_0$, the zero for 0 and the successor function for s, then we have a model for NAT and we get

Lemma 2 *The specification* NAT *is monomorphic and complete.*

Example 2 Next we want to extend Example 1 to a specification SEQ_N for sequences of natural numbers :

abstract data type SEQ_N **is**
extension of : NAT
basic sorts : $\underline{nat}$
new sort : $\underline{seq}$
constructors : $empty : \longrightarrow \underline{seq}$
$push : \underline{nat} \times \underline{seq} \longrightarrow \underline{seq}$
axioms : $\forall x \exists a(a = x \wedge \Diamond\Box a = empty \wedge$
$\Box(a = empty \vee \exists y \exists z(\bigcirc\Diamond a = z \wedge a = push(y, z))))$
$\forall x \forall y\ \ empty \neq push(x, y)$
$\forall x_1 \forall x_2 \forall y_1 \forall y_2(push(x_1, y_1) = push(x_2, y_2) \leftrightarrow x_1 = x_2 \wedge y_1 = y_2).$

The name for the operation *push* may be misleading, as until now no order of the elements is defined. Clearly we have a model with the set of sequences over $I\!N_0$ as carrier set, with the empty sequence as constant and with the usual push-operation, thus

Lemma 3 *The specification SEQ_N is monomorphic and complete.*

The main difference for the distinct data models of SEQ_N does not lie within the carrier sets which are all isomorphic, but within the different methods of access defined by additional operations. Later on we will see, that we can extend our specification SEQ_N by defining further operations. They then may depend on what kind of ouput behaviour is required as e.g. stacks or queues.

3 Extension by Definitions

3.1 Adding Predicate Symbols

Until now we have defined a new data type as a direct extension of an existing data type with free constructors as only operations. Normally in applications one wants more complex operations to access the data elements. These additional operations will be specified by definition, so that the underlying theory of a specification is not changed.

Definition 6 *Let $\mathbf{SP_1} = (\mathbf{\Sigma_1}, E_1)$ be a data specification. A specification $\mathbf{SP_2} = (\mathbf{\Sigma_2}, E_2)$ is called* extension by definitions, *if $\mathbf{F_2} = \mathbf{F_1} \cup \{p_1, ..., p_s\}$ for a set of new predicate symbols with arity $< i_1, ..., i_{n_j} >$, $n_j > 1$, for each p_j, and $E_2 = E_1 \cup E'$, where E' is the set of formulae*

$$\forall x_1 ... \forall x_{n_j} \; (p_j(x_1, ..., x_{n_j}) \leftrightarrow F_j(x_1, ...x_{n_j}))$$

for every predicate symbol p_j with x_r global variables of sort i_r and $F_j(x_1, ..., x_{n_j})$ is a formula from $\mathbf{TL}_{\mathbf{\Sigma_1}}$.

Now let us deal with the construction of data models for the extended signature.

Definition 7 *Let $\mathbf{SP_2}$ be an extension by definitions of $\mathbf{SP_1}$ with new predicate symbols $\{p_1, ..., p_s\}$. Let $\mathbf{A_1}$ be a data model of $\mathbf{SP_1}$ and $\mathbf{W}$ a corresponding valid state sequence. We define a $\mathbf{\Sigma_2}$-algebra $\mathbf{A_2} = (|\mathbf{A_2}|, \mathbf{F}_2^A)$ as* extension by definitions *of $\mathbf{A_1}$ with respect to $\mathbf{SP_2}$ by*

$$\begin{array}{l} |\mathbf{A_2}| = |\mathbf{A_1}| \\ \mathbf{F_2^A} = \mathbf{F_1^A} \cup \{p_1^{\mathbf{A_2}}, ..., p_s^{\mathbf{A_2}}\} \\ \textit{where for } 1 \leq j \leq s \; : \; p_j^{\mathbf{A_2}} \subseteq |\mathbf{A}|^{i_1} \times ... \times |\mathbf{A}|^{i_{n_j}} \\ \textit{with } (d_1, ..., d_{n_j}) \in p_j^{\mathbf{A_2}} \Leftrightarrow \mathbf{K_0}(F_j(x_1, ..., x_{n_j})) = \mathbf{t} \textit{ for} \\ \qquad\qquad\qquad\qquad \textit{every temporal structure } \mathbf{K} = (\mathbf{A_1}, \xi, \mathbf{W}) \\ \qquad\qquad\qquad\qquad \textit{with } \xi(x_r) = d_r, \; 1 \leq r \leq n_j. \end{array}$$

Lemma 4 *Let* $\mathbf{SP_2}$ *be an extension by definitions of* $\mathbf{SP_1}$. *A* $\mathbf{\Sigma_2}$*-algebra* $\mathbf{A_2}$ *is a data model for* $\mathbf{SP_2}$ *iff* $\mathbf{A_2}$ *is an extension by definitions with respect to* $\mathbf{SP_2}$ *of a data model for* $\mathbf{SP_1}$.

Proof : "$\Rightarrow$" Let $\mathbf{A_2}$ be a data model for $\mathbf{SP_2}$. Let $\mathbf{A_2} \setminus \{p_1^{\mathbf{A_2}}, ..., p_s^{\mathbf{A_2}}\}$ be the restriction without the new relations for $\{p_1, ...p_s\}$. Clearly $\mathbf{A_2} \setminus \{p_1^{\mathbf{A_2}}, ..., p_s^{\mathbf{A_2}}\}$ has to be a data model for $\mathbf{SP_1}$ and $\mathbf{A_2}$ is an extension by definitions of it.
"$\Leftarrow$" follows from the definition above. ♣

As by an extension by definitions nothing really "new" is added to the specification the following theorem holds.

Theorem 4 *Let* $\mathbf{SP_2}$ *be an extension by definitions of a monomorphic and complete specification* $\mathbf{SP_1}$, *then* $\mathbf{SP_2}$ *is monomorphic and complete.*

Proof : Let $\mathbf{A_1}$ and $\mathbf{A_1}'$ be two data models for $\mathbf{SP_1}$. From the assumptions there is an isomorphism φ from $\mathbf{A_1}$ to $\mathbf{A_1}'$. Let $\mathbf{A_2}$ and $\mathbf{A_2}'$ be respective extensions by definitions. Because of Lemma 4 it suffices to consider only such models. We only have to show that:

$$(d_1, ..., d_{n_j}) \in p_j^{\mathbf{A_2}} \Leftrightarrow (\varphi(t_1), ..., \varphi(t_{n_j})) \in p_j^{\mathbf{A_2}'}.$$

From the definition it holds that $(d_1, ..., d_{n_j}) \in p_j^{\mathbf{A_2}}$ iff $\mathbf{K} \models F_j(x_1, ..., x_{n_j})$ for every $K = (\mathbf{A_2}, \xi, \mathbf{W})$, where $\mathbf{W}$ is a valid state sequence and $\xi(x_r) = d_r$ for $1 \leq r \leq n_j$. By application of Lemma 1 and again by the definition we can prove the assertion.
Also the question of validity of a $\mathbf{SP_2}$-formula can be reduced to the question of validity of a $\mathbf{SP_1}$-formula and therefore the completeness follows. ♣

If we extend our specifications by new definitions, sometimes we want the new predicates to represent functions[3]. To express this, we will use the additional quantifier $\exists_1$ as abbrevation for

$$\exists_1 x \ F(x) \equiv \exists x \ F(x) \wedge \forall x \forall y \ (F(x) \wedge F(y) \rightarrow x = y).$$

So if we have an extension by definitions with new predicates $\{p_1, ..., p_s\}$ we say a predicate p_j *describes a total function* if the following formula is a theorem in the specification:

$$\forall x_1 ... \forall x_{n_j - 1} \exists_1 y \ p_j(x_1, ..., x_{n_j - 1}, y).$$

As notational convention we will use in the following the additional construct

defined operations :

to denote the new predicates.

[3]It is only for technical simplicity that we do not allow the extension by definition of new functions.

3.2 Natural Numbers

Now let us carry on with the data type NAT. In [Hil34] it is shown that the function of addition can not be defined in NAT in first-order logic. The same is valid for the multiplication in the so called Presburger arithmetic. While [Min88] has defined them in second-order logic, [Krö90] has shown that this is already possible in temporal logic. We will recall these last definitions and add some additional interesting predicates (or functions) as examples. Therefore we comprise several extension steps in the one following specification[4] :

abstract data type NAT_P **is**
extension of : NAT
basic sorts : $\underline{nat}$
defined operations : $add : \underline{nat} \times \underline{nat} \times \underline{nat}$
$mult : \underline{nat} \times \underline{nat} \times \underline{nat}$
$sub : \underline{nat} \times \underline{nat} \times \underline{nat}$
$leq : \underline{nat} \times \underline{nat}$
$fac : \underline{nat} \times \underline{nat}$
$div : \underline{nat} \times \underline{nat} \times \underline{nat}$

axioms :

$$add(x,y,z) \leftrightarrow \exists a \exists b (a = 0 \wedge b = y \wedge \Box((a = x \rightarrow z = b) \wedge (a \neq x \rightarrow \forall u \forall v (a = u \wedge b = v \rightarrow \bigcirc(a = s(u) \wedge b = s(v))))))$$

$$mult(x,y,z) \leftrightarrow \exists a \exists b (a = 0 \wedge b = 0 \wedge \Box((a = x \rightarrow z = b) \wedge (a \neq x \rightarrow \forall u \forall v (a = u \wedge b = v \rightarrow \bigcirc(a = s(u) \wedge add(y,v,b))))))$$

$$sub(x,y,z) \leftrightarrow \exists a \exists b (a = 0 \wedge b = x \wedge \Box((a = y \rightarrow z = b) \wedge (a \neq y \rightarrow \forall u \forall v (a = u \wedge b = v \rightarrow \bigcirc(a = s(u) \wedge (v \neq 0 \rightarrow s(b) = v) \wedge (v = 0 \rightarrow b = 0)))))))$$

$$leq(x,y) \leftrightarrow sub(x,y,0)$$

$$fac(x,y) \leftrightarrow \exists a \exists b (a = 0 \wedge b = 1 \wedge \Box((a = x \rightarrow b = y) \wedge (a \neq x \rightarrow \forall u \forall v (a = u \wedge b = v \rightarrow \bigcirc(a = s(u) \wedge (u = 0 \rightarrow b = v) \wedge (a \neq 0 \rightarrow mult(u,v,b)))))))$$

$$div(x,y,z) \leftrightarrow \exists a (a = 0 \wedge \Box \forall u \forall v ((a = v \wedge mult(s(v),y,u)) \rightarrow ((\neg leq(u,x) \rightarrow z = v)$$

[4] We will always omitt the universal quantifiers in front of the definition formulae.

$$\wedge\ (leq(u,x) \rightarrow \circ a = s(v))))$$
$$\wedge\ \Diamond\exists u(mult(s(a),y,u) \wedge leq(x,u))).$$

In the axioms a and b are local variables, all other variables are global. In [Krö90] it is shown that *add* and *mult* describe a total function, the same proof holds for sub and fac.
The predicate *div* describes a partial function, since div(x,0,z) is false for any x and z. We can observe at this example, that it is easy to define partial functions in our framework. They do not fulfill the requirement for total functions, but the following weaker formula :

$$\forall x_1 ... \forall x_{n_j-1}(\exists y\ p_j(x_1,...,x_{n_j-1},y) \rightarrow \exists_1 y\ p_j(x_1,...,x_{n_j-1},y)).$$

If we interpret our temporal logic formulae algorithmically, we may say that if a partial function gets an argument for which it is undefined, it loops forever and will never satisfy the termination criteria of the respective 'sometimes'-formula.

So now that we have defined some elementary functions for the natural numbers it may be interesting to find out what class of functions is definable in our framework. Thus we want to compare the expressivity of our approach with others, e.g. [Loe88] or [Klae84] who have used recursive or primitive recursive program schemes for constructive specifications.

Lemma 5 *Let* **SP** *be an extension by definition of NAT with a function symbol* $h : \underline{nat}^n \rightarrow \underline{nat}$. *It is possible to define in a monomorphically and complete extension of* **SP** *the* μ*-recursion* $\mu h : \underline{nat}^n \rightarrow \underline{nat}$ *for* h *such that*

$$\mu h(x_1,...,x_n) = min\{m \mid h(x_1,...,x_{n-1},m) = x_n\}$$

Proof : We will extend **SP** by a new predicate $\mu_h : \underline{nat}^{n+1}$ with the following definition:

$$\begin{aligned}\mu_h(x_1,...,x_n,z) \leftrightarrow \exists a\ (a = 0\ \wedge \\ \Box\ (h(x_1,...,x_{n-1},a) = x_n \rightarrow z = a\ \wedge \\ h(x_1,...,x_{n-1},a) \neq x_n \rightarrow \forall u\ (a = u \rightarrow \bigcirc a = s(u)))\ \wedge \\ \Diamond h(x_1,...,x_{n-1},a) = x_n)\end{aligned}$$

We will argue informally that μ_h modells μh.
It is easy to see that a takes as values the numbers 0, s(0), s(s(0)),... until the first number m is reached such that $h(x_1,...,x_{n-1},m) = x_n$. Thus m is the minimum. When the minimum is found the value of a is unique and thus μ_h fulfills the requirement of functionality. If the minimum does not exist then the predicate is false for any natural number, because the 'sometimes'-formula can't be satisfied. Thus we have specified a partial function. ♣

Lemma 6 *Let* **SP** *be an extension by definitions of NAT with total function symbols* $f : \underline{nat}^n \rightarrow \underline{nat}$ *and* $h : \underline{nat}^{n+2} \rightarrow \underline{nat}$. *It is possible to extend* **SP** *by defining the primitive recursive function* $g : \underline{nat}^{n+1} \rightarrow \underline{nat}$ *satisfying*

$g(x_1, ..., x_n, 0) = f(x_1, ..., x_n)$
$g(x_1, ..., x_n, m+1) = h(x_1, ..., x_n, m, g(x_1, ..., x_n, m))$

for all global variables x_i *and* m *of sort* $\underline{nat}$.

Proof : We will extend **SP** by a new predicate $g_{pr} : \underline{nat}^{n+2}$ as follows

$$\begin{aligned} g_{pr}(x_1, ..., x_n, m, z) \leftrightarrow \exists a \exists b\ (a = 0 \wedge b = f(x_1, ..., x_n) \wedge \\ \Box\ (a = m \rightarrow z = b \wedge \\ a \neq m \rightarrow \forall u \forall v\ (a = u \wedge b = v \rightarrow \\ \bigcirc(a = s(u) \wedge b = h(x_1, ..., x_n, u, v))))) \end{aligned}$$

First we want to prove that g_{pr} describes a total function. Let $\mathbf{A}$ be a data model for **SP** and $\mathbf{K} = (\mathbf{A}, \xi, \mathbf{W})$ a model for the axioms of **SP**. We construct a structure $\mathbf{K}' = (\mathbf{A}, \xi', \mathbf{W}')$ with $\xi \approx_z \xi'$ and $\mathbf{W} \approx_{a,b} \mathbf{W}'$ such that $\mathbf{K}' \models g_{pr}(x_1, ..., x_n, m, z)$ and the value $\xi'(z)$ is unique. We define the sequence of states inductively by :

$\eta'_0(a) = 0$ and $\eta'_0(b) = f^{\mathbf{A}}(\xi(x_1), ..., \xi(x_n)))$
and for all $i > 0$,
if $\eta'_i(a) = \xi(m)$ then $\eta'_j(a) = \eta'_i(a)$ and $\eta'_j(b) = \eta'_i(b)$ for every $j > i$,
else let $\eta'_{i+1}(a) = s^{\mathbf{A}}(\eta'_i(a))$ and
$\eta'_{i+1}(b) = h^{\mathbf{A}}(\xi(x_1), ..., \xi(x_n), \eta'_i(a), \eta'_i(b)) = d$, where d is a unique element of $|\mathbf{A}|$.

Because of Lemma 2 there is a smallest k such that $\eta'_k(a) = \xi(m)$. If we fix the unique value $\xi'(z) = \eta'_k(b)$ then we get the desired property.
Now we can define a total function $g : \underline{nat}^{n+1} \rightarrow \underline{nat}$ for which the above equations are valid by

$$g(x_1, ..., x_n, m) = z \leftrightarrow g_{pr}(x_1, ..., x_n, m, z).$$

To argue informally again, let us observe the consecutive values of the local variables a and b (for simplicity we will use $\tilde{x}$ instead of $x_1, ...x_n$)

a :	0	1	2	3
b :	$f(\tilde{x})$	$h(\tilde{x}, 0, f(\tilde{x}))$	$h(\tilde{x}, 1, h(\tilde{x}, 0, f(\tilde{x})))$	$h(\tilde{x}, 2, h(.....))$

and we see from the values of b which are the results of $g(\tilde{x}, a)$, that the above equations are satisfied. ♣

Hence for the question of expressivity we get the following result.

Theorem 5 *Let* **SP** *be an extension by definitions of NAT. Every recursive function in* **SP** *is definable.*

For example if we look at the specification NAT_P we see that *add* is defined by primitive recursion where f is the identity on $\underline{nat}$ and $h(x, y, z) = s(z)$.

3.3 Stacks and Queues

We have already defined the data type SEQ_N above. Now we want to show how we can extend the specification so that we get some more complex operations for the sort $\underline{seq}$.

abstract data type SEQU_N **is**
extension of : SEQ_N
basic sorts : $\underline{\mathrm{nat}}$, $\underline{\mathrm{seq}}$
defined operations :
$$isempty : \underline{seq}$$
$$in : \underline{seq} \times \underline{nat}$$
$$length : \underline{seq} \times \underline{nat}$$
$$reverse : \underline{seq} \times \underline{seq}$$
$$conc : \underline{seq} \times \underline{seq} \times \underline{seq}$$
axioms :
$$isempty(s) \leftrightarrow s = empty$$
$$in(s,x) \leftrightarrow \exists a(a = s \wedge \Box(\neg isempty(a) \wedge \forall u\ (a = u \rightarrow (\neg\exists v\ (u = push(x,v)) \rightarrow \circ\exists y(u = push(y,a))) \wedge (\exists v(u = push(x,v)) \rightarrow \circ\ a = u))))$$
$$length(s,m) \leftrightarrow \exists a \exists b(a = s \wedge\ b = 0 \wedge \Box((isempty(a) \rightarrow m = b) \wedge (\neg isempty(a) \rightarrow \forall u \forall v\ (a = u \wedge b = v \rightarrow \circ(\exists y(u = push(y,a)) \wedge\ b = s(v))))))$$
$$reverse(s,t) \leftrightarrow \exists a \exists b(a = s \wedge b = empty \wedge \Box((isempty(a) \rightarrow t = b) \wedge (\neg isempty(a) \rightarrow \forall u \forall v(a = u \wedge b = v \rightarrow \circ\exists x(u = push(x,a) \wedge b = push(x,v))))))$$
$$conc(s,t,r) \leftrightarrow \exists a \exists b(reverse(s,a) \wedge b = t \wedge \Box((isempty(a) \rightarrow r = b) \wedge (\neg isempty(a) \rightarrow \forall u \forall v(a = u \wedge b = v \rightarrow \circ\exists x(u = push(x,a) \wedge b = push(x,v))))))$$

Until now we have no operations which allow reading access to our objects of sort $\underline{seq}$. Therefore we can define different behaviours for them. For example we may allow to read any element for which the predicate in is true. More common are the two strategies which are known as LIFO or FIFO, which mean that we can only read the last input element or only the first input element. So next we will specify as extension of $SEQU_N$ the data type of stacks of natural numbers.

abstract data type $STACK$ **is**
extends : $SEQU_N$
basic sorts : $\underline{nat}, \underline{seq}$
defined operations : $top : \underline{seq} \times \underline{nat}$
$pop : \underline{seq} \times \underline{seq}$
axioms : $top(s,x) \leftrightarrow \exists r\ (s = push(x,r))$
$pop(s,t) \leftrightarrow \exists x\ (s = push(x,t))$

This data type is characterised by the property that one can always output only the last input element, which is a LIFO behaviour. Next we want to show that based on the same data type $SEQU_N$ we can specify by extension by definitions the data type of queues of natural numbers which follows a FIFO behaviour.

abstract data type $QUEUE$ **is**
extends : $SEQU_N$
basic sorts : $\underline{nat}, \underline{seq}$
defined operations : $first : \underline{seq} \times \underline{nat}$
$rest : \underline{seq} \times \underline{seq}$
axioms : $first(q,x) \leftrightarrow \exists a(a = q \wedge \neg isempty(a) \wedge$
$\Box(\neg isempty(a) \rightarrow$
$\forall u\ (a = u \rightarrow (\neg \exists y\ (u = push(y, empty)) \rightarrow$
$\circ \exists y\ (u = push(y,a))) \wedge$
$\forall y\ (u = push(y, empty)) \rightarrow y = x))))$
$rest(q,r) \leftrightarrow \exists a \exists b(a = q \wedge b = empty \wedge \neg isempty(a) \wedge$
$\Box(\neg isempty(a) \rightarrow$
$\forall u \forall v(a = u \wedge b = v \rightarrow$
$(\exists x(u = push(x, empty)) \rightarrow reverse(b,r)) \wedge$
$(\neg \exists x(u = push(x, empty)) \rightarrow$
$\circ \exists x(u = push(x,a) \wedge b = push(x,v))))))$

Note that $first$ and $rest$ describe partial functions which are not defined for the empty sequence in the first argument.
We also may now define the abstract data type $DEQUE$, which allows access from both ends of the sequence, as extension of $STACK$ with e.g. the definition of $first$:

$first(q,x) \leftrightarrow \forall r\ (reverse(q,r) \rightarrow top(r,x))$.

4 Expressiveness

Up to now we have built specifications by using constructors such that there is an isomorphism between the term algebra and every other data model. This is a strong requirement

and we will not always find a data model for our set of axioms, or, the other way round, given a specific data model it may be impossible to find sufficient free constructor functions. For example let us regard the sets of natural numbers. To build them from the empty set we need a constructor function to insert elements in a set. This may be the operation $push$ like in the data typ SEQ_N. But then we are in trouble, because we would need an axiom like

$$\forall x \forall s \ (push(x, push(x, s)) = push(x, s))$$

which conflicts with the (ID) axiom, since we might deduce then that every set is equal to the empty set which is certainly not a desired property and is in contradiction to the (DIS)-axiom. The above shows that the (ID) axiom is too strong because it requires equality. There are different approaches in the literature to solve this problem, like e.g. the semantic constructions in [Loe88]. We will again follow the approach of [Min88] which provides a solution for a further class of data models. Later on we will see, that then we are as expressive as the classical approach.

Let $\mathbf{SP_2}$ be a direct extension of $\mathbf{SP_1}$ with new sort $k+1$, such that all the non-constant constructor functions f_j have arity $< i_1, ..., i_{n_j}, k+1 >$ with $n_j \geq 1$ and $1 \leq i_1, ..., i_{n_j} \leq k$, so we have no arguments of sort $k+1$. In this case the set of new constant symbols may be empty. Now we will replace the axiom scheme (ID) by the new axiom scheme

$$\text{(ID')} \ \forall(*)(f_j(x_1, ..., x_{n_j}) = f_j(y_1, ..., y_{n_j}) \leftrightarrow A_j(x_1, ..., x_{n_j}, y_1, ..., y_{n_j})),$$

where each A_j is a formula which satisfies the following conditions :

$$\forall(*) \ A_j(x_1, ..., x_{n_j}, x_1, ..., x_{n_j})$$
$$\forall(*) \ A_j(x_1, ..., x_{n_j}, y_1, ..., y_{n_j}) \rightarrow A_j(y_1, ..., y_{n_j}, x_1, ..., x_{n_j})$$
$$\forall(*) \ A_j(x_1, ..., x_{n_j}, y_1, ..., y_{n_j}) \wedge A_j(y_1, ..., y_{n_j}, z_1, ..., z_{n_j}) \rightarrow A_j(x_1, ..., x_{n_j}, z_1, ..., z_{n_j}).$$

As we have now special kind of constructor functions we can also simplify the (CONS)-axiom to :

$$\text{(CONS')} \ \forall x(\bigvee_{i=1}^{l} x = c_i \vee \bigvee_{j=1}^{m} \exists y_1 ... \exists y_{n_j} \ x = f_j(y_1, ..., y_{n_j})).$$

In our new axiomsets no temporal operator appears anymore. This means that for this restricted class of data types the classical first-order logic is sufficient to specify monomorphism and completeness.

Theorem 6 *Let $\mathbf{SP_2} = (\mathbf{\Sigma}, E)$ be a direct extension of a monomorphic $\mathbf{SP_1}$, such that the new sort $k+1$ appears not in the arguments of the constructors. If E^{k+1} is the set of the respective instances of (CONS'), ,(DIS) and (ID') then $\mathbf{SP_2}$ is monomorphic.*

Proof : Let $\mathbf{A_1}$ be a data model for $\mathbf{SP_1}$ and the $\mathbf{\Sigma}$-algebra $\mathbf{A_2}$ be defined in the same way like in the proof for Theorem 2[5]. With the A_j we can define a binary relation R_j on $|\mathbf{A_2}|^{k+1}$ by :

[5]It appears that in this case the definition of the carrier set for the new sort $k+1$ is not inductiv. That is why we do not need temporal logic for the (CONS)-axiom.

$c_i R_j c_k \leftrightarrow c_i = c_k$
$f_j(x_1, ..., x_{n_j}) R_j f_j(y_1, ..., y_{n_j}) \leftrightarrow A_j(x_1, ..., x_{n_j}, y_1, ..., y_{n_j})$
$f_i(x_1, ..., x_{n_i}) R_j f_i(y_1, ..., y_{n_i}) \leftrightarrow f_i(x_1, ..., x_{n_i}) = f_i(y_1, ..., y_{n_i})$, for $i \neq j$.

From the conditions on the A_j it follows that this is an equivalence relation. With Q_r denoting the identity relation on $|\mathbf{A_2}|^r$ for $1 \leq r \leq k$ we can construct a congruence relation $\theta_j = \{Q_r : 1 \leq r \leq k\} \cup \{R_j\}$ on $|\mathbf{A_2}|$. Then also $\theta = \bigcup_{j=1}^{m} \theta_j$ is a congruence relation. Herewith we get a Σ-algebra, the quotient term algebra $\mathbf{A_2}/\theta$, by taking the congruence classes of respective sort of θ in $\mathbf{A_2}$ as carrier sets, i.e. $|\mathbf{A_2}/\theta|^{\mathbf{i}} = \{[d]_\theta, d \in |\mathbf{A_2}|^{\mathbf{i}}\}$ and defining the operations $f^{\mathbf{A_2}/\theta}([d_1]_\theta, ..., [d_n]_\theta) := [f^{\mathbf{A_2}}(d_1, ..., d_n)]_\theta$ for every $f \in \mathbf{F}$ with arity $< i_1, ..., i_n >$. It is easy to see that the quotient algebra $\mathbf{A_2}/\theta$ satisfies the axioms (CONS'),(DIS) and (ID'). Thus $\mathbf{A_2}/\theta$ is a model for $\mathbf{SP_2}$. By the same isomorphism as defined in Theorem 2 we can prove that all other models of $\mathbf{SP_2}$ are isomorphic to it. ♣

In addition it is obviously that with the new axiomsets nothing is changed with respect to the completeness of a specification.

So now we are ready to specify the abstract data type SET in our framework. We will join therefore the direct extension of $SEQU_N$ and the following extension by definitions in one specification.

abstract data type SET **is**
extension of : $SEQU_N$
basic sorts : $\underline{nat}$, $\underline{seq}$
new sort : $\underline{set}$
constructors : $makeset : \underline{seq} \rightarrow \underline{set}$
defined operations : $isempty : \underline{set}$
$insert : \underline{set} \times \underline{nat} \times \underline{set}$
$delete : \underline{set} \times \underline{nat} \times \underline{set}$
$member : \underline{set} \times \underline{nat}$
$subset : \underline{set} \times \underline{set}$
axioms : $\forall s^{set} \exists q^{seq}\ (s^{set} = makeset(q^{seq}))$
$\forall p^{seq} \forall q^{seq} (makeset(p^{seq}) = makeset(q^{seq}) \leftrightarrow$
$\forall n^{nat} (in(p^{seq}, n^{nat}) \leftrightarrow in(q^{seq}, n^{nat})))$

$isempty(s) \leftrightarrow s = makeset(empty)$
$insert(s, n, t) \leftrightarrow \forall q (s = makeset(q) \rightarrow t = makeset(push(q, n)))$
$delete(s, n, t) \leftrightarrow \exists a \exists b (a = s \wedge isempty(b) \wedge \Diamond isempty(a) \wedge$
$\Box((isempty(a) \rightarrow t = b) \wedge$
$(\neg isempty(a) \rightarrow$
$\exists v \exists u \exists m (insert(u, m, a) \wedge b = v \wedge$
$\bigcirc(a = u \wedge (m = n \rightarrow b = v) \wedge$

$$(m \neq n \rightarrow insert(v,m,b))))))$$
$$member(s,n) \leftrightarrow \forall q(s = makeset(q) \rightarrow in(q,n))$$
$$subset(s,t) \leftrightarrow \exists a(a = s \wedge \Diamond isempty(a) \wedge$$
$$\Box\exists v\exists u\exists n(a = v \wedge insert(u,n,v) \wedge$$
$$(\neg member(t,n) \rightarrow \circ a = v) \wedge$$
$$(member(t,n) \rightarrow \circ a = u))))$$

where a and b are local variables of type $\underline{set}$, p and q are global variables of type $\underline{seq}$, m and n are global variables of type $\underline{nat}$ and the rest are global variables of type $\underline{set}$.

Next we want to show that our conditions placed on the specifications are not really restricting. The proof uses similiar constructions like in the work of [Merz91].

Theorem 7 *Let* $\mathbf{A}$ *be a* Σ*-algebra. If there exists a specification for* $\mathbf{A}$ *with a finite set of equations in the initial semantic framework, then in our constructive framework with temporal logic formulae there exists a monomorphic and complete specification for* $\mathbf{A}$ *using direct extension and extension by definitions.*

Proof : Let $\mathbf{A}$ be a $\mathbf{\Sigma}$-algebra. For simplicity we may assume $\mathbf{\Sigma}$ to be one-sorted. Thus $\mathbf{A}$ consists of a carrier set $|\mathbf{A}|$ and a set of operations $\mathbf{F^A} = \{f_1^{\mathbf{A}}, ..., f_m^{\mathbf{A}}\}$. Let $SPEC = (\mathbf{\Sigma}, E)$ be the specification, where $E = \{s_i = t_i | 1 \leq i \leq n\}$ is a finite set of equations, such that every initial algebra of $SPEC$ is isomorph to $\mathbf{A}$.
Now let $\mathbf{\Sigma} = (\{s\}, \{c_1, ..., c_k, f_1, ..., f_m\})$ where each f_j, for $1 \leq j \leq m$, has arity $\underbrace{< s, ..., s}_{n_j - times}, s >$. We define a signatur $\mathbf{\Sigma'} = (\{s'\}, \{c_1', ..., c_k', f_1', ..., f_m'\})$ where each operation symbol f_i' has arity $\underbrace{< s', ..., s'}_{n_j - times}, s' >$.
From each equation $s_i = t_i$ in E we construct the terms s_i' and t_i' by replacing the symbols of $\mathbf{\Sigma}$ with the primed symbols of $\mathbf{\Sigma'}$ and letting the variables run over the sort s'.
Now let us choose every constant and operation symbol of $\mathbf{\Sigma'}$ as a constructor. Thus, as shown in Theorem 2 and Theorem 3, with the respective instances of the axioms (CONS),(DIS) and (ID) we get a monomorphic and complete specification $\mathbf{SP_1}$. A data model for $\mathbf{SP_1}$ is the term algebra of $\mathbf{\Sigma'}$.
In the next step we extend the specification $\mathbf{SP_1}$ by the definition of a new predicate $eqp_E : s' \times s'$. Intuitively the predicate is true for two primed terms iff it is possible to deduce the equality of the unprimed terms from the axioms of E in the equational calculus. The definition of eqp_E is as follows :

$$eqp_E(x,y) \leftrightarrow \exists a\exists b(a = x \wedge b = y \wedge \Diamond\Box(a = b \vee (\bigvee_{i=1}^{n} a = s_i' \wedge b = t_i')) \wedge$$
$$\Box(a = b \vee (\bigvee_{i=1}^{n} a = s_i' \wedge b = t_i') \vee$$
$$\exists p\exists q(\ a = p \wedge b = q \wedge$$
$$(\circ\Diamond(a = q \wedge b = p) \vee$$
$$\exists z(\circ\Diamond(a = p \wedge b = z) \wedge \circ\Diamond(a = z \wedge b = q)) \vee$$

$$\bigvee_{j=1}^{m} \exists x_1 ... \exists x_{n_j} \exists y_1 ... \exists y_{n_j}$$
$$(\bigwedge_{r=1}^{n_j} (\circ\Diamond(a = x_r \wedge b = y_r) \wedge p = f'_j(x_1, ..., x_{n_j})$$
$$\wedge q = f'_j(y_1, ..., y_{n_j})))))$$

If we look at the definition of eqp_E we see that in the first state the values of the local variables a and b are equal to x and y. In every state the values of a and b are identical or they are equal to the left and right side of one of the primed axioms or their values may be obtained by applying one of the rules of symmetry, transitivity or substitutivity to some succeeding values of themselves. Thus we have for one state that the equality of the values for a and b is deducible if it is in each following state. As from a certain state on the values of a and b are constant and deducibly equal, the predicate eqp_E is true for all x and y for which the equality is deducible from primed E in the equational calculus. The other way round it is clear that if the equality of two terms of sort s' is deducible from the primed axioms of E, then there exists a deduction, such that the reverse of it may be simulated by a state sequence of two local variables, beginning in the first state with the values of the two terms and from one state on the local variables are permanently equal to the left and right side of one of the premises of the deduction. Thus we have

$$eqp_E(x', y') \Leftrightarrow E \vdash_{EQ} x = y$$

where EQ denotes a complete deduction system for the equational calculus (see e.g. [Ehr85] or [Wir90]).
Obviously eqp_E is an equivalence relation. The new specification with the predicate eqp_E is denoted by $\mathbf{SP_2}$. Now we define a direct extension $\mathbf{SP_3}$ of $\mathbf{SP_2}$ with new sort s and the only free constructor $make_s : s' \rightarrow s$. As axioms we take the respective instances of the formulae in Theorem 6 :

$$\forall x \exists x'(x = make_s(x'))$$
$$\forall x' \forall y'(make_s(x') = make_s(y') \leftrightarrow eqp_E(x', y'))$$

If we further extend $\mathbf{SP_3}$ by additional operations f_j which are defined by

$$f_j(x_1, ..., x_{n_j}) = y \leftrightarrow \forall z'_1 ... \forall z'_{n_j}(\bigwedge_{i=1}^{n_j} x_i = make_s(z'_i) \rightarrow y = make_s(f'_j(z'_1, ..., z'_{n_j})))$$

we get a monomorphic and complete specification $\mathbf{SP_4}$.
It is easy to see that for every data model $\mathbf{A_4}$ of $\mathbf{SP_4}$ its restriction $\mathbf{A_4}|_{\mathbf{\Sigma}}$ is isomorphic to the quotient term algebra $\mathbf{T_\Sigma}/\equiv_E$, where $\equiv_E$ is the congruence relation induced by E. And as $\mathbf{T_\Sigma}/\equiv_E$ is initial for $SPEC$, for every data model $\mathbf{A_4}$ of $\mathbf{SP_4}$ its restriction $\mathbf{A_4}|_{\mathbf{\Sigma}}$ is isomorphic to $\mathbf{A}$. Thus we have a complete and monomorphic specification for $\mathbf{A}$. ♣

5 Computability of Algebras

In [Wir90] it is shown, that in first-order logic all models of a monomorphic specification are computable and that the class of initial (terminal) models of a specification is semi-computable (co-semicomputable). Similiar to Theorem 7 we will now show that in our approach we may specify semicomputable and co-semicomputable algebras monomorphically. This demonstrates the greater expressive power of temporal logic for specification of data types.
To prove our next theorem we first have to recall some notions about computability of algebras (see also [Wir90]). For a signature $\mathbf{\Sigma}$ a $\mathbf{\Sigma}$-algebra is called a *number algebra* if its carrier sets are recursive subsets of the set $I\!N_0$ of the natural numbers. A *coordinatization* of a $\mathbf{\Sigma}$-algebra A is a pair $< \mathbf{C}, \alpha >$, where $\mathbf{C}$ is a number algebra of signature $\mathbf{\Sigma}$ and $\alpha : \mathbf{C} \to A$ an epimorphism. For any sensible signature $\mathbf{\Sigma}$ (i.e. which admits at least one ground term for each sort) exists a bijective coordinatization $< \mathbf{C_\Sigma}, v >$ of the term algebra, i.e. with an isomorphism $v : \mathbf{C_\Sigma} \to T_\mathbf{\Sigma}$ and a fixed number algebra $\mathbf{C_\Sigma}$.
For any $\mathbf{\Sigma}$-algebra $\mathbf{A}$, the associated $\mathbf{\Sigma}$-congruence $\sim^\mathbf{A} \subseteq \mathbf{T_\Sigma} \times \mathbf{T_\Sigma}$ is defined by :

$$t \sim^\mathbf{A} t' \Leftrightarrow t^\mathbf{A} = t'^\mathbf{A}.$$

Then $\mathbf{A}$ is isomorphic to the quotient term algebra $\mathbf{T_\Sigma} / \sim^\mathbf{A}$.

To characterize the classes of $\mathbf{\Sigma}$-algebras we will use a theorem from [Wir90].

Theorem 8 *Let $\mathbf{A}$ be a $\mathbf{\Sigma}$-algebra with associated $\mathbf{\Sigma}$-congruence $\sim^\mathbf{A}$. Then the following properties are equivalent :*
1.) $\mathbf{A}$ is computable, semicomputable or co-semicomputable respectively
2.) $\sim^\mathbf{A}$ is recursive, recursively enumerable or co-recursively enumerable respectively.

The computability notions of algebras are invariant under isomorphism.

We now can state an assertion about the specifiability of number algebras

Lemma 7 *Let $\mathbf{C}$ be a one-sorted recursive number algebra with carrier set $I\!N_0$ and signature $\mathbf{\Sigma} =< \{\underline{nat}\}, \mathbf{F} >$. Then $\mathbf{C}$ has a monomorphic and complete specification which is an extension by definitions of* NAT *with additional function symbols $\mathbf{F} \cup \mathbf{F}'$, where $\mathbf{F}'$ is a set of auxiliary function symbols.*

Proof : The algebra $\mathbf{C}$ consists of the carrier-set $I\!N_0$ and of recursive functions $f^\mathbf{C} : I\!N_0^k \to I\!N_0$ for every $f : \underline{nat}^k \to \underline{nat}$ in $\mathbf{F}$. In Theorem 5 we have shown that every recursive function is definable in an extension by definitions of NAT. For every recursive function symbol $f \in \mathbf{F}$ let A_f be the defining formula for the recursive function $f^\mathbf{C}$. In A_f we may need auxiliary function symbols for the definition of f. Then we denote by E_f the set of formulae, which consists of A_f and the defining formulae for the auxiliary function symbols occuring in A_f. Also let $\mathbf{F}_f$ be the union of f with the set of the auxiliary function symbols occuring in A_f. Then we define

abstract data type $\mathbf{SP_C}$ is
extension of : NAT
basic sorts : $\underline{nat}$
defined operations $\bigcup\{\mathbf{F}_f | f \in \mathbf{F}\}$
axioms : $\bigcup\{E_f | f \in \mathbf{F}\}$.

As NAT is monomorphic and complete then because of Theorem 4 $\mathbf{SP_C}$ is monomorphic and complete. For every model $\mathbf{A}$ of $\mathbf{SP_C}$ the $\mathbf{\Sigma}$-restriction $\mathbf{A}|_{\mathbf{\Sigma}}$ is isomorphic to $\mathbf{C}$. ♣

Lemma 8 *Let $\mathbf{\Sigma} = <\{s\}, \mathbf{F}>$ be a one-sorted signature and let $\mathbf{F}^N = \{f^N : \underline{nat}^k \to \underline{nat} | f : s^k \to s \in \mathbf{F}\}$ be the set of function symbols from $\mathbf{F}$ with an upper index N and ranging over $\underline{nat}$. There exists a monomorphic and complete specification $\mathbf{SP}$ with signature $\mathbf{\Sigma_{SP}}$ which is an extension by definitions of* NAT *with additional function symbols $\mathbf{F}^N \cup \mathbf{F}'$ where $\mathbf{F}'$ is a set of auxilliary function symbols, such that the recursive number algebra $\mathbf{C_\Sigma}$ which is associated with $T_{\mathbf{\Sigma}}$ is embedded in any model $\mathbf{A}$ of $\mathbf{SP}$, i.e. $\mathbf{C_\Sigma} \cong \mathbf{A}|_{in}$ where $in : \mathbf{\Sigma} \to \mathbf{\Sigma_{SP}}$ is defined by $in(s) = \underline{nat}$ and $in(f) = f^N$ for each $f \in \mathbf{F}$.*

Proof : $\mathbf{C_\Sigma}$ is a one-sorted recursive number algebra. Thus it suffices to apply Lemma 7 and to take the appropriate specification $\mathbf{SP}_{\mathbf{C_\Sigma}}$ for $\mathbf{SP}$. ♣

Theorem 9 *Let $\mathbf{\Sigma}$ be a finite signature and let $\mathbf{A}$ be a $\mathbf{\Sigma}$-algebra. If $\mathbf{A}$ is semicomputable or co-semicomputable, then in temporal logic there exists a monomorphic and complete specification $\mathbf{SP}$ such that for every data model $\mathbf{A_{SP}}$ of $\mathbf{SP}$ the $\mathbf{\Sigma}$-restriction $\mathbf{A_{SP}}|_{\mathbf{\Sigma}}$ is isomorphic to $\mathbf{A}$.*

Proof : W.l.o.g. we may assume that $\mathbf{\Sigma}$ is one-sorted, i.e. $\mathbf{\Sigma} = <\{s\}, \mathbf{F}>$. For $\mathbf{\Sigma}$ exists a bijective coordinatization $<\mathbf{C_\Sigma}, v>$. According to Lemma 8 $\mathbf{C_\Sigma}$ can be embedded in a monomorphic and complete specification $\mathbf{SP}_{\mathbf{C_\Sigma}}$.
Let at first $\mathbf{A}$ be a semicomputable $\mathbf{\Sigma}$-algebra. Then $\sim^{\mathbf{A}}$ is recursively enumerable, i.e. the relation $\sim^v$ defined by

$$i \sim^v j \Leftrightarrow v(i) \sim^{\mathbf{A}} v(j) \text{ for } i, j \in I\!N_0$$

is recursively enumerable. Thus we can choose primitive recursive functions $g^{\mathbf{A}}$, $h^{\mathbf{A}} : I\!N_0 \to I\!N_0$ to enumerate it, so that $\sim^v = \{(g^{\mathbf{A}}(x), h^{\mathbf{A}}(x)) | x \in I\!N_0\}$. Hence we have that $\{v(g^{\mathbf{A}}(x)) = v(h^{\mathbf{A}}(x)) | x \in I\!N_0\}$ is the set of ground equations which hold in $\mathbf{T_\Sigma}/\sim^{\mathbf{A}}$.
Let g, h : $\underline{nat} \to \underline{nat}$ be the function symbols corresponding to $g^{\mathbf{A}}$ and $h^{\mathbf{A}}$. According to Theorem 4 and Lemma 7 $g^{\mathbf{A}}$ and $h^{\mathbf{A}}$ admit a finite set $E_g \cup E_h$ of defining axioms, where the axioms for the auxilary functions occuring in the definitions of $g^{\mathbf{A}}$ and $h^{\mathbf{A}}$ are included. We then may state a specification

abstract data type $\mathbf{SP_1}$ is
extension of : $\mathbf{SP_{C_\Sigma}}$
basic sort : $\underline{nat}$
defined operations : $\mathbf{F}_g \cup \mathbf{F}_h$
axioms : $E_g \cup E_h$

which is monomorphic and complete.
Next we define a direct extension of $\mathbf{SP_1}$ with new sort s' and a set $\mathbf{F}$' of constructor functions which is the primed version of $\mathbf{F}$ with respective arity for each symbol in $\mathbf{F}$' (see proof for Theorem 5). For this new specification $\mathbf{SP_2}$ we take for the axiom set E_2 the respective instances of the axiom schemes (CONS), (DIS) and (ID) from Theorem 2. Then $\mathbf{SP_2}$ is monomorphic and complete.
Now we extend $\mathbf{SP_2}$ by the definition of a new predicate $\nu : \underline{nat} \times s'$ with the following axiom :

$$\begin{aligned}\nu(n,d) \leftrightarrow \exists a \exists b (a = n \wedge b = d \wedge \\ (\textstyle\bigvee_{i=1}^{l} \Diamond\Box(a = c_i^N \wedge b = c_i')) \wedge \\ \Box \textstyle\bigwedge_{i=1}^{l} ((a = c_i^N \rightarrow b = c_i') \wedge \\ \textstyle\bigwedge_{j=1}^{m} \exists x_1 ... \exists x_{n_j} (a = f_j^N(x_1, ..., x_{n_j}) \rightarrow \\ \exists y_1 ... \exists y_{n_j} (b = f_j'(y_1, ..., y_{n_j}) \wedge \\ \textstyle\bigwedge_{r=1}^{n_j} \bigcirc\Diamond(a = x_r \wedge b = y_r)))))\end{aligned}$$

The predicate ν in this specification $\mathbf{SP_3}$ defines the bijective mapping $v : I\!N_0 \rightarrow \mathbf{T_\Sigma}$. Similiar like in the proof for Theorem 6 we can introduce a single constructor function $make_s : s' \rightarrow s$ which maps the objects of the term algebra to the objects of the intended quotient term algebra. Then we get a further specification

abstract data type SP_4 **is**
extension of : SP_3
basic sorts : $\underline{nat}, s'$
new sort : s
constructors : $make_s : s' \rightarrow s$
axioms : $\forall x \exists x' (x = make_s(x'))$
$\forall x' \forall y' \; (make_s(x') = make_s(y') \leftrightarrow$
$\exists z^{nat} \; ((\nu(g(z), x') \wedge \nu(h(z), y')) \vee (\nu(h(z), x') \wedge \nu(g(z), y')))$

That the right side of the equivalence in the second axiom defines an equivalence relation follows then from the properties of the functions $g^{\mathbf{A}}$, $h^{\mathbf{A}}$ and v.
In the last step we extend our specification to define the functions from $\mathbf{F}$ by the following set of axioms

$$\begin{aligned}\forall(*)(f(x_1, ..., x_n) = y \leftrightarrow \forall z_1' ... \forall z_n' \; (\textstyle\bigwedge_{i=1}^{n} x_i = make_s(z_i') \rightarrow \\ y = make_s(f_j'(z_1', ..., z_n'))))\end{aligned}$$

with which we obtain the specification **SP**. As in every step the monomorphism and the completeness of the specification is preserved and for every data model $\mathbf{A_5}$ of **SP** the $\mathbf{\Sigma}$-restriction is isomorphic to $\mathbf{T_\Sigma}/\sim^{\mathbf{A}}$, we have a monomorphic and complete specification for **A**.

Now let **A** be a co-semicomputable algebra. Then the complement $\not\sim^{\mathbf{A}}$ of $\sim^{\mathbf{A}}$ is recursively enumerable, i.e. the relation

$$i \not\sim^{v} j \Leftrightarrow v(i) \not\sim^{\mathbf{A}} v(j) \text{ for } i,j \in I\!N_0$$

is recursively enumerable. Again then we can choose primitive recursive functions $g^{\mathbf{A}}$, $h^{\mathbf{A}} : I\!N_0 \to I\!N_0$ so that $\{v(g^{\mathbf{A}}(x)) \neq v(h^{\mathbf{A}}(x) | x \in I\!N_0\}$ is the set of ground inequations which are valid in $\mathbf{T_\Sigma}/\sim^{\mathbf{A}}$.
Just as above we then can construct a specification $\mathbf{SP_3}$ with the definitions of the functions $g^{\mathbf{A}}$, $h^{\mathbf{A}}$ and ν, which again we extend by a function $make_s : s' \to s$ where we now substitute the second axiom by the following :

$$\forall x' \forall y' \; (make_s(x') = make_s(y') \leftrightarrow \\ \neg\exists z \; ((\nu(g(z), x') \wedge \nu(h(z), y')) \vee (\nu(h(z), x') \wedge \nu(g(z), y'))))$$

The right side of the equivalence then describes an equivalence relation.
By extending this specification similiar to the proof above with the definitions of the functions from **F** we get a monomorphic and complete specification for which the $\mathbf{\Sigma}$-restriction of every data model is isomorphic to $\mathbf{T_\Sigma}/\sim^{\mathbf{A}}$ and therefore to **A**. ♣

6 Conclusion

We have shown that with our constructive specification method with temporal logic formulae as axioms, we are at least as expressive as equational specifications with initial semantics. By an appropriate extension of the predicate eqp_E it may be easily seen, that our proof above also holds for specifications using horn clauses. A major drawback of our approach may be that proving in our incomplete logic will be much more complex than in the classical case. But we have shown that we are able to state general applicable axiom schemes, which allow us to specify the monomorphism and the completeness of a data type without using any meta assumptions. And we have proven that we may specify semicomputable and co-semicomputable data types monomorphically. In this sense the expressive power of temporal logic for specification of data types is stronger than that of classical first-order logic. We also have seen, that temporal logic is powerful enough to define additional, even partial, operations without changing the underlying set of objects.
Our specifications combine two interesting features. First they are abstract, using axioms written in a formal logical language. But some of these formulae may also be interpreted algorithmically. It should not be too difficult to extract programs in an imperative

language from our temporal logic axioms. And as research proceeds in executing directly temporal logic formulae, our specifications may also be regarded yet as an implementation in a high level programming language .
Temporal logic is often used for the description of parallel behaviours. In an integrated approach of [Krö87b] it is suggested to use temporal logic as logical language for the specification of the data types and as description language for valid state sequences of parallel systems. The here presented work shall be a further step in the analysis of the potentiality of such an approach, where arbitrary temporal logic formulae may be used for the specification of data types.

References

[Ehr85] H.Ehrig, B.Mahr : Fundamentals of Algebraic Specifications 1, Springer Verlag (1985)

[Hil34] D.Hilbert, P.Bernays : Grundlagen der Mathematik, Springer Verlag (1934)

[Klae84] H.A.Klaeren : A constructive method for abstract algebraic software specification, *Theoretical Computer Science* **30** (1984) 139-204

[Krö87a] F.Kröger : Temporal Logic of Programs, Springer Verlag (1987)

[Krö87b] F.Kröger : Abstract Modules : Combining algebraic and temporal logic specification means, *Techniques et Science Informatiques* **6** (1987) 559-573

[Krö90] F.Kröger : On the interpretability of arithmetic in temporal logic, *Theoretical Computer Science* **73** (1990) 47-60

[Loe87] J.Loeckx : Algorithmic Specifications: A Constructive Specification Method for Abstract Data Types, *ACM Trans. on Progr. Languages and Systems* **9** (1987) 646-685

[Merz91] S.Merz : Characterizing Initial Models of Abstract Data Types Using Temporal Logic, Ludwig Maximilians Universität München, Institut für Informatik, Report 91/02 (1991)

[Min88] Z.Ming-Hua : A Second Order Theory of Data Types, *Acta Informatica* **25** (1988) 283-303

[Mir87] G.Mirkowska, A.Salwicki : Algorithmic Logic, D.Reindel Publishing Company (1987)

[Sza88] A.Szalas : Towards the temporal approach to abstract data types, *Fundamenta Informaticae* **11** (1988) 49-64

[Wir90] M.Wirsing : Algebraic Specification, in : Handbook of Theoretical Computer Science II, North Holland, Amsterdam (1990)