BRICS RS-96-52 Hintermeier et al.: Combining Algebraic and Set-Theoretic Specifications

BRICS

Basic Research in Computer Science

Combining Algebraic and Set-Theoretic Specifications

(Extended Version)

Claus Hintermeier
Hélène Kirchner
Peter D. Mosses

BRICS Report Series **RS-96-52**

ISSN 0909-0878 **December 1996**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through World Wide Web and anonymous FTP:

http://www.brics.dk/
ftp://ftp.brics.dk/
This document in subdirectory RS/96/52/

Combining Algebraic and Set-Theoretic Specifications. (Extended version)*

Claus Hintermeier, Hélène Kirchner
CRIN-CNRS/INRIA Lorraine
B.P. 239
F-54506 Vandœuvre-lès-Nancy Cedex
email: `hinterme,hkirchne@loria.fr`

Peter D. Mosses
BRICS[†], University of Aarhus
Ny Munkegade, bldg. 540
DK-8000 Aarhus C
email: `pdmosses@brics.dk`

Abstract

Specification frameworks such as B and Z provide power sets and cartesian products as built-in type constructors, and employ a rich notation for defining (among other things) abstract data types using formulae of predicate logic and lambda-notation. In contrast, the so-called algebraic specification frameworks often limit the type structure to sort constants and first-order functionalities, and restrict formulae to (conditional) equations. Here, we propose an intermediate framework where algebraic specifications are enriched with a set-theoretic type structure, but formulae remain in the logic of equational Horn clauses. This combines an expressive yet modest specification notation with simple semantics and tractable proof theory.

1 Introduction

As is well known, there are two main schools of thought regarding the formal specification of abstract data types: the model-oriented [1, 18, 3], and the property-oriented [5, 20]. Let us briefly recall the main features of these two approaches.

Model-oriented specifications. The emphasis is on specifying data types as set-theoretic structures (products, power sets, etc.), the operations of the data types then being defined as particular functions on these structures. For example, to specify a simple abstract data type of sequences, one defines the type as a set of functions, represents a sequence $\langle x_1, \ldots, x_n \rangle$ by the function mapping i to x_i for each $i \in \{1, \ldots, n\}$, and defines concatenation using function

*This work was partially supported by COMPASS (ESPRIT Basic Research Working Group 6112).

†Centre for Basic Research in Computer Science

abstraction and application. The underlying logic for reasoning about such a specification is a powerful higher-order logic, e.g., based on ZF set theory.

Property-oriented specifications. Here one generally tries to *avoid* choosing an explicit representation: types are left abstract as so-called sorts—sometimes equipped with a subsort inclusion relation, but otherwise unstructured. The operations are specified by axioms that relate them to each other, often including the main intended algebraic properties. For example, sequences could be specified by axioms asserting that concatenation is associative, with the empty sequence as unit. The underlying logic is often a modest Horn-clause fragment of equational first-order logic—supplemented by an induction rule when dealing with initial algebra semantics rather than loose semantics.

Combinations. In practice, some model-oriented specification languages (such as Z) do allow types to be left abstract (or 'given'), with the operations on them specified by axioms. Moreover, the use of auxiliary ('hidden') sorts and operations in property-oriented specifications can give these a model-oriented flavour. There are also some wide-spectrum languages (e.g., RSL, Spectrum) which encompass both approaches, allowing model- and property-oriented specifications to be mixed together.

It seems to us that both the model- and property-oriented approaches have their advantages *and* disadvantages. In particular, we regard the restriction to Horn-clause logic in the latter as beneficial, since not only are the consequences of a specification much more obvious than in full higher-order logic, but also automated reasoning and prototyping are feasible. The restriction of types to unstructured sorts in property-oriented specifications, however, we regard as a definite disadvantage.

This has led us to investigate an intermediate or hybrid approach, combining the better features of the model- and property-oriented approaches:

- Types include abstract types (of individuals) as well as product, power set, and function types.
- Operations may be higher-order and partial.
- The only built-in relations are equality, set membership and the existential predicate (the latter merely abbreviates an equality).
- Formulae are restricted to Horn-clauses (no disjunction, no explicit negation, variables are universally quantified).
- Models have quite straightforward set-theoretic foundations.
- Specifications have initial models (when consistent).
- Specifications are amenable to prototyping and reasoning using rewriting and saturation techniques.

We hope that the illustrations given in Section 2 will convince the reader that this framework is rather expressive, at least for specifying abstract data types. The foundations laid in Section 3 are reasonably simple, and Section 4 demonstrates the tractability of the logic. Finally, Section 5 concludes by comparing the proposed framework with related approaches.

This paper is an extended version of [7]. We present proofs of the results, and improve some technical details in the definitions.

2 Illustrations of Specifications

Before we take a closer look at the foundations of our proposed framework, let us illustrate its expressiveness, which approaches that of model-oriented frameworks such as B and Z, and far exceeds that of conventional algebraic specifications. The examples given correspond to fragments of the standard Mathematical Toolkit for Z: abstract data types of numbers and sequences.

We have chosen to format our specifications in the style normally used for Z in the literature, because the symbols here mostly have roughly the same intended interpretation as that indicated in the Z standard. However, one should bear in mind that our specifications are *not* actually in Z itself, and the formal details of interpretations here are different. Moreover, we want the possibility of initial algebra semantics for our specifications. For those readers not familiar with Z, a few extra words of explanation of the examples will be given below.

NaturalNumbers

$$
\begin{array}{l}
\mathbb{N} : \mathbb{P}(\mathbb{I}) \\
0 : \mathbb{N} \\
succ : \mathbb{N} \nrightarrow \mathbb{N} \\
\_ \leq \_, \_ < \_ : \mathbb{N} \leftrightarrow \mathbb{N} \\
\hline
\forall\, x, y : \mathbb{N} \bullet \quad
\begin{array}{l}
0 \leq x \\
x \leq y \Rightarrow succ(x) \leq succ(y) \\
0 < succ(x) \\
x < y \Rightarrow succ(x) < succ(y)
\end{array}
\end{array}
$$

All constants used in the axioms in the lower part of the specification are declared, with their types, in the upper part. The type $\mathbb{I}$ is that of all individuals, and $\mathbb{P}$ forms the type of sets with elements from its argument type, thus $\mathbb{N}$ is specified to be a set of individuals. Sets may be used to indicate types in declarations: since $\mathbb{N} : \mathbb{P}(\mathbb{I})$ is a set of individuals, $0 : \mathbb{N}$ declares the type of 0 to be $\mathbb{I}$. Moreover, for any constant c declared to have type T, there is an implicit axiom $c \in T$, where the type T is now interpreted as a set.

The remaining constants declared above are all first-order: $succ$ has the type of a partial function on $\mathbb{N}$, and $\leq$ and $<$ are typed as binary relations. (We could have declared $succ$ to have total function type $\mathbb{N} \rightarrow \mathbb{N}$, giving a slightly stronger theory, but we prefer to keep this example minimal to facilitate the discussion of its operationalization in Section 4.)

The axioms above consist of two unconditional and two conditional Horn-clauses—we leave their conjunction implicit. The universal quantification of the variables used in the clauses is given explicitly, so that all notation used has been declared, and to avoid the need for type inference.

Our next example is a generic data type of sequences. X is a type variable, determining the type of elements in the sequences, and the constants declared have polymorphic types: they may be used with X instantiated by any ground (monomorphic) type, e.g., $\mathbb{I} \times \mathbb{I}$. The reference to the *NaturalNumbers* specification makes the notation declared there available locally, for use in the specification of the *length* operation. The constant *seq* is supposed to denote a function that maps any subset S of X to the set $seq(S)$ of sequences whose elements are all in S.

The type specified for $\_ \frown \_$ illustrates the use of product types when specifying operations of more than one argument. Note also that we use the place-holder symbol '$\_$' to indicate where the arguments go when an operation is applied: we allow not only ordinary prefix notation, but also infix, postfix, outfix, and general mixfix notation for applications. The place-holder may be omitted in prefix symbols.

GenericSequences $[X]$

NaturalNumbers

$seq : \mathbb{P}(X) \mathrel{+\!\!\!\rightarrow} \mathbb{P}(\mathbb{I})$

$\langle\,\rangle : seq(X)$

$\langle \_ \rangle : X \mathrel{+\!\!\!\rightarrow} seq(X)$

$\_ \frown \_ : seq(X) \times seq(X) \mathrel{+\!\!\!\rightarrow} seq(X)$

$length : seq(X) \mathrel{+\!\!\!\rightarrow} \mathbb{N}$

$$
\begin{aligned}
&\forall\, x : X \bullet \forall\, s, t, u : seq(X) \bullet \\
&\qquad \langle\,\rangle \frown s = s \\
&\qquad s \frown \langle\,\rangle = s \\
&\qquad s \frown (t \frown u) = (s \frown t) \frown u \\
&\qquad length(\langle\,\rangle) = 0 \\
&\qquad length(\langle x \rangle \frown s) = succ(length(s))
\end{aligned}
$$

Let us consider now a higher-order example. The notation $t \downarrow$ is read as "t exists" (or "t is defined").

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad MappingSequences\ [X, Y] \qquad\qquad\qquad\qquad}} \\
GenericSequences \\
mapseq : (X \twoheadrightarrow Y) \twoheadrightarrow (seq(X) \twoheadrightarrow seq(Y)) \\
\hline
\forall x : X \bullet \forall s, t : seq(X) \bullet \forall f : X \twoheadrightarrow Y \bullet \\
\quad mapseq(f)(\langle\rangle) = \langle\rangle \\
\quad f(x) \downarrow \Rightarrow mapseq(f)(\langle x \rangle) = \langle f(x) \rangle \\
\quad mapseq(f)(s) \downarrow \wedge mapseq(f)(t) \downarrow \Rightarrow \\
\qquad mapseq(f)(s \frown t) = mapseq(f)(s) \frown mapseq(f)(t) \\
\hline
\end{array}
$$

In this example we have to be a bit careful about existence: equations are interpreted as 'existential', so $x = y$ asserts the existence of both x and y. Since f ranges over partial functions, $f(x)$ may not exist for some x. If f were restricted to being a total function, we could replace *all* the $\twoheadrightarrow$'s by $\rightarrow$'s above, and drop the existence conditions from the axioms altogether.

3 Foundations

Here we provide the formal foundations for our specification language. For perspicuity, we treat first a simpler kernel language, giving syntax, semantics, proof rules, and various results; in particular, we show how to decide whether a term is well-typed, as soon as each constant and variable has a type. Models for specifications in the kernel language are defined in a set theoretic framework. Sound proof rules are given for this class of models, which has an initial model. The construction of this initial model is a key point to prove completeness of deduction, more precisely that any atomic formula valid in all models is deducible with these proof rules. At the end of this section we define the syntax of our complete language, and indicate how it can be reduced to the kernel language. First, we make some general remarks about the novel aspects of our work.

Compared to conventional algebraic specification languages, the type system of our language is very rich, being close to that of Z. We follow Z in interpreting types as sets, and in allowing them to be used as values in terms too. However, we take a slightly *intensional* view of sets: two sets that have the same values as members are not necessarily equal! This is achieved by what amounts to a labelling of sets, provided by a *choice function* (denoted by *choose*) which gives a unique element for any set given as argument. This provides the key to the completeness of our logic.

The motivation for not using extensional equality on sets is twofold. First, an axiom in a specification could have an equality between sets as a condition. If two differently-specified sets (say, subsets of the natural numbers) happened to have the same members, unexpected consequences might follow. By insisting that two sets are only regarded as equal when this follows by *algebraic* reasoning, rather than by membership equivalence, we remove this (admittedly minor) danger. Second, for tractability and operationalization of reasoning, we wish to remain in Horn-clause logic (with equality), and it is well-kown that extensionality cannot be axiomatized in (finitary) Horn-clause logic. Adding a choice function, freely interpreted, allows formulating a deduction rule for an extensional equality on sets with non-standard elements. Our treatment of higher-order operations is similarly intensional: two operations do not get equated just because they give the same results on the same arguments. We claim that this kind of intensional equality of sets and functions is necessary to keep our framework truly 'algebraic'. Other frameworks for higher-order algebraic specification, e.g., that of [11], provide extensionality and term-models, but the proof theory seems less tractable.

For expressing types, we propose, in addition to the constant $\mathbb{I}$ for the built-in type of individuals[1] and variables, several type constructors: partial power set operator $\mathbb{P}$ and cartesian product $\times$, and partial function space constructor $\nrightarrow$ and relation space constructor $\leftrightarrow$. Furthermore, we introduce the term-generated subset constructor $\mathbb{T}$ that extracts from any set its term-generated part; this is needed by our choice to consider also non-standard (i.e. non term-generated) models. Finally, for technical reasons, we include in our type system a universal type $\mathbb{U}$.

In Z, the function space $S \nrightarrow T$ and relation space $S \leftrightarrow T$ are both interpreted as subtypes of $\mathbb{P}(S \times T)$. This may seem rather natural, but it has some unfortunate pragmatic consequences in connection with overloading and subtype polymorphism: specifying $f : S \nrightarrow T$ restricts the graph of f to be a subset of $S \times T$, preventing f from being extended to give results on arguments not in S! For example, in the Z framework specifying $succ : \mathbb{N} \nrightarrow \mathbb{N}$ implies $succ \subseteq \mathbb{N} \times \mathbb{N}$, which prevents an extension to $succ : \mathbb{Z} \nrightarrow \mathbb{Z}$ with $succ(-1) = 0$. (According to the Z Draft Standard, $\mathbb{N}$ is *not* a type, but merely a subset of $\mathbb{Z}$. The constant $succ$ thus gets a unique interpretation on $\mathbb{Z}$, and $(-1, 0) \notin \mathbb{N} \times \mathbb{N}$.) Similarly, in Z a relation $p : S \leftrightarrow T$ cannot be extended to other types, so specifying e.g. $\_ \leq \_ : \mathbb{N} \times \mathbb{N}$ prevents overloading with $\_ \leq \_ : \mathrm{seq}(\mathbb{N}) \times \mathrm{seq}(\mathbb{N})$.

Here, in contrast to Z, our interpretation of $S \nrightarrow T$ is as a set of partial functions which always (when defined) give results in T *when applied to arguments in S!* Thus specifying $f : S \nrightarrow T$ does not say anything at all about what might result when f is applied to arguments outside S. An axiom $f \in S \nrightarrow T$ here corresponds to $S \lhd f \in S \nrightarrow T$ in Z, where $S \lhd f$ denotes the restriction of f to domain S (but our type assertion $f : S \nrightarrow T$ appears to have no exact counterpart in Z). Similarly, we interpret $S \leftrightarrow T$ as a set of binary relations that may relate other pairs than those in $S \times T$.

By this means, we are able to avoid a significant danger of accidental inconsistency in specifications, and permit the use of subtype polymorphism and overloading, as common in some algebraic specification frameworks. Notice that our change of interpretation does *not* prevent type-checking: $f(s)$ is only considered type-correct when $f : S \nrightarrow T$ and $s : S$ hold for some S and T.

[1]For simplicity in this paper, we do not consider further 'given' types of individuals.

3.1 Syntax

A *presentation* $\mathcal{P}$ in our kernel language consists of a set of declarations of typed constants (including $\mathbb{I}, \mathbb{P}, \mathbb{T}$, *choose*, $\times, \nrightarrow, \leftrightarrow$) and variables, together with a set of Horn-clauses built from terms over the declared items using three built-in predicates: equality $=$, membership $\in$ and existence $\downarrow$. The fixed notation is indicated by the following BNF-like grammar:

$$
\begin{array}{lcl}
\mathit{CLAUSE} & ::= & \mathit{ATOMS} \Rightarrow \mathit{ATOM} \;\big|\; \mathit{ATOMS} \Rightarrow \\
\mathit{ATOMS} & ::= & \mathit{ATOM} \wedge \mathit{ATOMS} \;\big|\; \mathit{ATOM} \\
\mathit{ATOM} & ::= & \mathit{TERM} = \mathit{TERM} \;\big|\; \mathit{TERM} \in \mathit{TERM} \;\big|\; \mathit{TERM}\downarrow \\
\mathit{TERM} & ::= & \mathit{CONST} \;\big|\; \mathit{VAR} \;\big|\; \mathit{TERM}(\mathit{TERM}) \;\big|\; \mathit{TERM}, \mathit{TERM} \\
\mathit{CONST} & ::= & \mathbb{I} \;\big|\; \mathbb{P}\_ \;\big|\; \mathbb{T}\_ \;\big|\; \mathit{choose}\_ \;\big|\; \_\times\_ \;\big|\; \_\nrightarrow\_ \;\big|\; \_\leftrightarrow\_ \;\big|\; \ldots \\
\mathit{VAR} & ::= & \ldots
\end{array}
$$

where the '...'s indicate declared constants and variables. Place-holders '_' in constant symbols indicate that these are to denote functions, and that applications are to be written concretely with the argument(s) in the position of the place-holder(s). Parentheses are allowed for disambiguating the grouping. Thus the application of $\mathbb{P}\_$ to $\mathbb{I}$ may be written $\mathbb{P}\,\mathbb{I}$ or $\mathbb{P}(\mathbb{I})$, and that of $\_\times\_$ to $(\mathbb{I}, \mathbb{I})$ as $\mathbb{I} \times \mathbb{I}$. The syntax for pairs $\mathit{TERM}, \mathit{TERM}$ and applications of product $\mathit{TERM} \times \mathit{TERM}$ is assumed to be left-associative. Note that a list of ATOMS may be empty.

Types are denoted by *type terms* generated by the following grammar:

$$
\begin{array}{lcl}
\mathit{TYPE} & ::= & \mathit{VAR} \;\big|\; \mathbb{I} \;\big|\; \mathbb{U} \;\big|\; \mathit{TYPE} \times \mathit{TYPE} \\
& \big| & \mathbb{P}(\mathit{TYPE}) \;\big|\; \mathit{TYPE} \nrightarrow \mathit{TYPE} \;\big|\; \mathit{TYPE} \leftrightarrow \mathit{TYPE} \\
\mathit{RTYPE} & ::= & \mathbb{T}(\mathit{VAR}) \;\big|\; \mathbb{T}(\mathbb{I}) \;\big|\; \mathbb{T}(\mathbb{U}) \;\big|\; \mathbb{T}(\mathit{TYPE} \times \mathit{TYPE}) \\
\mathit{STYPE} & ::= & \mathbb{T}(\mathbb{P}(\mathit{TYPE})) \;\big|\; \mathbb{T}(\mathit{TYPE} \nrightarrow \mathit{TYPE}) \;\big|\; \mathbb{T}(\mathit{TYPE} \leftrightarrow \mathit{TYPE}) \\
\mathit{TTYPE} & ::= & \mathit{RTYPE} \;\big|\; \mathit{STYPE}
\end{array}
$$

Type terms will be interpreted as sets of values (when defined), with variables VAR indicating polymorphism. We shall regard TYPE, RTYPE, STYPE, and TTYPE as *kinds*, writing $T : K$ to assert that T is a well-formed type term of kind K. Intuitively, values of types in TTYPE are generated by 'standard' terms, excluding the use of *choose*; the types in STYPE are simply those that are interpreted as sets of sets, so that *choose* may be used on their elements. For example, the types $\mathbb{I} \times \mathbb{P}(\mathbb{I})$, $\mathbb{P}(\mathbb{I}) \nrightarrow \mathbb{P}(\mathbb{I})$ and $\mathbb{P}(\mathbb{P}(\mathbb{I}))$ are of kind TYPE and $\mathbb{T}(\mathbb{I} \times \mathbb{P}(\mathbb{I}) \times \mathbb{P}(\mathbb{P}(\mathbb{I})))$ is of kind TTYPE.

Type terms are partially ordered by a subtype inclusion ordering $\leq$. The cartesian product constructor is monotone in both arguments, as are $\mathbb{P}$ and $\mathbb{T}$ in their only argument. The function space constructor is anti-monotone in the first argument and monotone in the second. The relation space constructor is anti-monotone in both arguments, just as if it were a boolean function. So the subtype inclusion ordering $\leq$ on type terms satisfies the following assertions, where S, T, V, W range over well-formed type terms of kind $TYPE$ only:

$$
\begin{array}{l}
S \leq V \Rightarrow \mathbb{P}(S) \leq \mathbb{P}(V) \\
S \leq V \wedge T \leq W \Rightarrow S \times T \leq V \times W \\
V \leq S \wedge T \leq W \Rightarrow S \nrightarrow T \leq V \nrightarrow W \\
V \leq S \wedge W \leq T \Rightarrow S \leftrightarrow T \leq V \leftrightarrow W \\
S \leq \mathbb{U} \\
V \nrightarrow W \leq \mathbb{P}(\mathbb{U} \times \mathbb{U}) \\
V \leftrightarrow W \leq \mathbb{P}(\mathbb{U} \times \mathbb{U}) \\
S \leq V \Rightarrow \mathbb{T}(S) \leq \mathbb{T}(V) \\
\mathbb{T}(S) \leq S
\end{array}
$$

together with reflexivity and transitivity of $\leq$. Note that the omission of the inclusions $S \nrightarrow T \leq S \leftrightarrow T \leq \mathbb{P}(S \times T)$ is deliberate, to allow overloading, as discussed at the start of Section 3. The inclusion $V \nrightarrow W \leq \mathbb{P}(\mathbb{U} \times \mathbb{U})$ reflects the fact that a partial function (graph) is a set of pairs, without restriction on the components of the pairs; similarly for binary relations in $V \leftrightarrow W \leq \mathbb{P}(\mathbb{U} \times \mathbb{U})$.

Proposition 3.1 *The validity of formulae* $\exists \ldots \forall \ldots (S_1 \leq T_1 \wedge \ldots \wedge S_n \leq T_n)$ *where* S_i, T_i *for* $i \in \{1, \ldots, n\}$ *are of kind TYPE or TTYPE, is decidable.*

Proof: Given a type inclusion problem of the form $\exists\forall(S_1 \leq T_1 \wedge \ldots \wedge S_n \leq T_n)$, we start with the replacement of universally quantified type variables by new constants of the same type. Let the so obtained purely existentially quantified problem C be $\exists(S'_1 \leq T'_1 \wedge \ldots \wedge S'_n \leq T'_n)$.

The conjunction is then rewritten using the following set of rewrite rules, closely corresponding to subtyping assertions above, and taking into account in the last additional rule that an inequation with the $\mathbb{T}$ operator, that can only occur at the top of type terms, can be weakened.

$$
\begin{array}{rcl}
\mathbb{P}(S) \leq \mathbb{P}(V) & \rightarrow & S \leq V \\
S \times T \leq V \times W & \rightarrow & S \leq V \wedge T \leq W \\
S \nrightarrow T \leq V \nrightarrow W & \rightarrow & V \leq S \wedge T \leq W \\
S \leftrightarrow T \leq V \leftrightarrow W & \rightarrow & V \leq S \wedge W \leq T \\
S \leq \mathbb{U} & \rightarrow & \mathit{true} \\
V \nrightarrow W \leq \mathbb{P}(\mathbb{U} \times \mathbb{U}) & \rightarrow & \mathit{true} \\
V \leftrightarrow W \leq \mathbb{P}(\mathbb{U} \times \mathbb{U}) & \rightarrow & \mathit{true} \\
\mathbb{T}(S) \leq \mathbb{T}(V) & \rightarrow & S \leq V \\
\mathbb{T}(S) \leq S & \rightarrow & \mathit{true} \\
\mathbb{T}(T) \leq S & \rightarrow & T \leq S \\
 & & \textit{if } S \notin \mathit{VAR},\ S \textit{ not of kind } \mathit{TTYPE}.
\end{array}
$$

This system terminates since each rewrite step strictly decreases the number of symbols $\mathbb{P}, \times, \nrightarrow, \leftrightarrow, \mathbb{U}, \mathbb{T}$ in the conjunction. After simplifying with respect to the usual properties of conjunction, the result C' is either true or a conjunction of inequations in which one side is a variable or the constant $\mathbb{I}$. Clearly, if $C \longrightarrow^* C'$, then $C' \Rightarrow C$ logically. In particular if $C \longrightarrow^* \mathit{true}$, then C is true. Otherwise, the resulting set of inequations has a solution if and only if the left and right-hand sides of the inequations are unifiable with respect to kinds (which is decidable in this order-sorted regular signature of type terms of kinds $\{\mathit{TYPE}, \mathit{RTYPE}, \mathit{STYPE}, \mathit{TTYPE}\}$). $\square$

Finally, we define type membership for terms (potentially) denoting values, writing $s : T$ to assert that s is a well-formed term of type T. Let X, Y be type variables of kind TYPE, and let f, s, t range over arbitrary terms. Then we assert:

$$
\begin{array}{ll}
s : S \wedge S \leq T \Rightarrow s : T & \\
f : \mathbb{T}(S \nrightarrow T) \wedge s : \mathbb{T}(S) \Rightarrow f(s) : \mathbb{T}(T) & \\
s : \mathbb{T}(S) \wedge t : \mathbb{T}(T) \Rightarrow (s, t) : \mathbb{T}(S \times T) & \\
\mathbb{I} : \mathbb{T}(\mathbb{P}(\mathbb{I})) & \\
\mathbb{P} : \mathbb{T}(\mathbb{P}(X) \nrightarrow \mathbb{P}(\mathbb{P}(X))) & \mathbb{T} : \mathbb{T}(\mathbb{P}(X) \nrightarrow \mathbb{T}(\mathbb{P}(X))) \\
\mathit{choose} : \mathbb{T}(\mathbb{P}(X)) \nrightarrow X & \_ \times \_ : \mathbb{T}(\mathbb{P}(X) \times \mathbb{P}(Y) \nrightarrow \mathbb{P}(X \times Y)) \\
\mathit{choose} : \mathbb{T}(X \nrightarrow Y) \nrightarrow (X \times Y) & \_ \nrightarrow \_ : \mathbb{T}(\mathbb{P}(X) \times \mathbb{P}(Y) \nrightarrow \mathbb{P}(X \nrightarrow Y)) \\
\mathit{choose} : \mathbb{T}(X \leftrightarrow Y) \nrightarrow (X \times Y) & \_ \leftrightarrow \_ : \mathbb{T}(\mathbb{P}(X) \times \mathbb{P}(Y) \nrightarrow \mathbb{P}(X \leftrightarrow Y))
\end{array}
$$

Observe that in general, constants are declared to have types of kind TTYPE, the exception being choose.

Proposition 3.2 *Assuming that the declarations in a presentation provide types for each constant and variable used, it is decidable to check that each term is well-typed.*

Proof: From constant and variable types, it is possible to determine in a bottom-up process a type for any term built from these constants and variables with the pairing and function application operators. Because of subtyping and overloading, a term may have several types.

In order to type $f(a)$ knowing $a : A$ and $f : F$, we solve

$$\exists\, S, T \quad (F \leq \mathbb{T}(S \nrightarrow T) \ \wedge \ A \leq \mathbb{T}(S)).$$

Let μ be a solution. Then $f(a) : \mathbb{T}(\mu(T))$.

In order to type (a_1, a_2) knowing $a_1 : A_1$ and $a_2 : A_2$, we solve

$$\exists\, S, T \quad (A_1 \leq \mathbb{T}(S) \ \wedge \ A_2 \leq \mathbb{T}(T)).$$

Let μ be a solution. Then $(a_1, a_2) : \mathbb{T}(\mu(S) \times \mu(T))$. $\square$

For example, if $\mathbb{N} : \mathbb{T}(\mathbb{P}(\mathbb{I}))$, then we can form the term $\mathbb{P}(\mathbb{N})$, which is of type $\mathbb{T}(\mathbb{P}(\mathbb{P}(\mathbb{I})))$ since $\mathbb{P}\_ : \mathbb{T}(\mathbb{P}(X) \nrightarrow \mathbb{P}(\mathbb{P}(X)))$ and the system

$$\exists S, T \quad (\mathbb{T}(\mathbb{P}(X) \nrightarrow \mathbb{P}(\mathbb{P}(X))) \leq \mathbb{T}(S \nrightarrow T) \quad \wedge \quad \mathbb{T}(\mathbb{P}(\mathbb{I})) \leq \mathbb{T}(S))$$

has a solution with $X = \mathbb{I}$, $S = \mathbb{P}(\mathbb{I}))$ and $T = \mathbb{P}(\mathbb{P}(\mathbb{I}))$.

In the following sections, for an arbitrary presentation $\mathcal{P}$, we write $\mathbf{F}, \mathbf{X}$ for the declared type constructors and variables (with their respective kinds), $\mathcal{F}, \mathcal{X}$ for the remaining constants and variables (with their respective types), $\overline{\Sigma}$ for $(\mathcal{F}, \mathbf{F}, \mathbf{X})$, and Σ for $\overline{\Sigma}$ without the constant *choose* (which plays a special role in our framework). A $\overline{\Sigma}$-presentation is given by a signature $(\overline{\Sigma}, \mathcal{X})$ and a set of $(\overline{\Sigma}, \mathcal{X})$-Horn-clauses. In the following, $(\Sigma, \mathcal{X})$-terms are called standard terms and $(\overline{\Sigma}, \mathcal{X})$-terms are called non-standard terms.

3.2 Semantics

The semantics for our formulae are structural set theoretic models. We therefore start with the definition of an interpretation for the type constructors (from $\mathbf{F}$) and other constants (from $\mathcal{F}$). These interpretations are not given separately, since both can be used in terms. Remark that we use a classical notion of set in the definition, as e.g. in [19].

Note furthermore that typing and membership are not to be confused: except for declared constants, deriving $t : T$ does not imply that $t \in T$ holds. This difference comes from the use of partial functions. E.g., $div(1, 0)$ may be of type $\mathbb{I}$, but may fail to exist, i.e. $div(1, 0)$ does not denote a member of the set $\mathbb{I}$.

Definition 3.3 *A* standard set-theoretic Σ-interpretation I *is a pair* $(U_I, .^I)$ *of a universe* U_I *and an interpretation function* $.^I$*, such that:*

- U_I *is a set of atomic objects (individuals and pairs) and sets (the special constant* $\mathbb{U}$ *is* not *interpreted as an element of* U_I*, to avoid foundational problems—conceptually, it may be interpreted as the entire universe* U_I*);*
- $\mathbb{I}^I$ *is a set of individuals;*
- $\mathbb{P}^I$ *is a partial function such that* $\mathbb{P}^I(S)$*, when defined, is a subset of the ordinary powerset of* S*;*
- $\times^I$ *is a partial function yielding, when defined, the cartesian product of its arguments;*
- $\nrightarrow^I$ *is a partial function yielding, when defined, a set of (graphs of) partial functions such that when* $g \in S \nrightarrow^I T$ *then* $(s, t) \in g$ *and* $s \in S$ *imply* $t \in T$*;*
- $\leftrightarrow^I$ *is a partial function yielding, when defined, a set of (graphs of) binary relations;*
- $(\_, \_)^I$ *is a partial function yielding, when defined, the pair of its arguments;*

- *$f(s)^I$ is t when f is a function graph and $(s,t) \in f$;*

- *for all constants c in $\mathcal{F} \cup \mathbf{F}$ (other than $\mathbb{U}$, $\mathbb{T}$ and choose), $c^I \in \alpha^I(T)$ when $c : T$ for all $(\mathbf{X}, I)$-variable assignments α (i.e. a mapping from type variables to $(\mathbf{F}, \varnothing)$-type term interpretations), and where α^I is the natural extension of α to terms under interpretation $.^I$. When α is $\varnothing$, $\alpha^I(T)$ is written as T^I.*

Note that $\mathbb{T}^I$ is left unrestricted above. Now we relax the notion of interpretation to allow non-standard elements (i.e. not denoted by a Σ-term) that represent (but are not themselves) sets. These non-standard elements may be regarded as 'labels' that distinguish between sets which include the same standard elements.

Definition 3.4 *A* (non-standard) $\overline{\Sigma}$-interpretation *I is a standard set-theoretic Σ-interpretation with the following differences:*

- *The sets in $\mathbb{P}^I(S)$ may include non-standard elements that represent (but are not themselves) subsets of S;*

- *$\mathbb{T}^I$ is a partial function such that, when defined, $\mathbb{T}^I(S)$ is the Σ-term-generated subset of S (excluding all non-standard elements);*

- *$choose^I$ is a partial function which gives a unique arbitrary element from any non-empty set $S \in U_I$, and is otherwise undefined.*

A $\overline{\Sigma}$-interpretation where $\mathbb{I}^I$ is a set of $\overline{\Sigma}$-terms is called a $\overline{\Sigma}$-term interpretation.

Let us illustrate Definition 3.4 with an example.

Example 3.5 Let $\mathcal{F} = \{s, 0, \_ \leq \_\}$ with $0 : \mathbb{I}$, $s : \mathbb{I} \twoheadrightarrow \mathbb{I}$, $\_ \leq \_ : \mathbb{I} \leftrightarrow \mathbb{I}$. Let furthermore *Nat* and *Even* be of type $\mathbb{P}(\mathbb{I})$. Then I defined as follows is a $\overline{\Sigma}$-interpretation:

- U_I is a set containing all natural numbers, the set of even natural numbers, the set of all natural numbers, the graph of the successor function, the graph of the less or equal relation, and all pairs of natural numbers. Moreover, $\mathbb{U}_I$ has to contain the denotations of all the built-in constants (other than $\mathbb{U}$), i.e., $\mathbb{I}$, $\mathbb{P}$, $\times$, $\twoheadrightarrow$, $\leftrightarrow$, $\mathbb{T}$, and *choose*.

- $\mathbb{I}^I$ is the set of natural numbers,

- $\mathbb{P}^I$ maps $\mathbb{I}^I$ to $\{Nat^I, Even^I\}$ and is undefined for all other arguments.

- $\times^I$ is the empty partial function graph;

- $\twoheadrightarrow^I$ maps $(\mathbb{I}^I, \mathbb{I}^I)$ to the singleton containing the graph of the successor function only; it is undefined for all other arguments.

- $\leftrightarrow^I$ maps $(\mathbb{I}^I, \mathbb{I}^I)$ to the singleton containing the graph of the less or equal relation; it is undefined for all other arguments.

- $\_(\_)^I$ is defined as required for $\overline{\Sigma}$-interpretations.
- $(\_,\_)^I$ maps any two natural numbers to their pair, and is otherwise undefined;
- 0^I is zero, s^I is the (graph of the) successor function on natural numbers, $\leq^I$ is less or equal on natural numbers, Nat^I is the set of natural numbers and $Even^I$ is the set of even natural numbers.
- $\mathbb{T}^I$ is the identity on $\mathbb{I}^I$, $\mathbb{P}^I(\mathbb{I}^I)$, $\mathbb{I}^I \nrightarrow^I \mathbb{I}^I$, and $\mathbb{I}^I \leftrightarrow^I \mathbb{I}^I$; and all their cartesian products built with $\times$; it is undefined otherwise.
- The choice function $choose^I$ gives[2] zero for $Even^I$, one for Nat^I (and hence for $\mathbb{I}^I$), the graph of the successor function for $(\mathbb{I} \nrightarrow \mathbb{I})^I$ and the less or equal relation for $(\mathbb{I} \leftrightarrow \mathbb{I})^I$.

Hence, we get for example, $0^I \in \mathbb{I}^I$, $Nat^I \in \mathbb{P}^I(\mathbb{I}^I))$, $Even^I \in \mathbb{P}^I(\mathbb{I}^I))$, and $s(0)^I =$ one as expected. Remark that $\mathbb{T}(\mathbb{P}(\mathbb{I}))^I$ is the set containing only the set of natural numbers and the set of even natural numbers but not the set of odd natural numbers.

However, since we also consider non-standard interpretations, any interpretation J defined as I, except that $\mathbb{I}^J$ is the set of (positive and negative) integers, is another $\overline{\Sigma}$-interpretation. Then, $\mathbb{I}^I$ and $\mathbb{I}^J$ differ, but all other interpretations remain unchanged, since the term generated part of I and J is equal. Here $\mathbb{T}^J$ on $\mathbb{I}^J$ cuts out the negative integers.

Definition 3.6 *Let I, J be $\overline{\Sigma}$-interpretations.*
A $\overline{\Sigma}$-homomorphism *$h : I \longrightarrow J$ is a (total) function $h : U_I \longrightarrow U_J$ satisfying:*

- *for all $(\mathbf{F}, \varnothing)$-terms T, $t \in T^I$ implies $h(t) \in T^J$;*
- *for all constants $c \in \mathcal{F} \cup \mathbf{F}$ (apart from $\mathbb{U}$), $h(c^I) = c^J$;*
- *for all $(\mathbf{F}, \varnothing)$-terms T_1, T_2 of kind $TTYPE$, for all $t_1 \in T_1^I, t_2 \in T_2^I$, $h((t_1, t_2)^I) = (h(t_1), h(t_2))^J$ whenever (t_1, t_2) exists;*
- *for all $(\mathbf{F}, \varnothing)$-terms T, T' of kind $TYPE$, for all $t_2 \in \mathbb{T}(T)^I$ and for all $t_1 \in \mathbb{T}(T \nrightarrow T')^I$, $h(t_1(t_2)^I) = h(t_1)(h(t_2))^J$ whenever $t_1(t_2)$ exists.*

Definition 3.7 *A $(\mathbf{X} \cup \mathcal{X}, I)$*-variable assignment *$\alpha$ is a mapping from variables in $\mathbf{X} \cup \mathcal{X}$ to $(\mathbf{F}, \varnothing)$-type terms and elements of $\mathbb{I}^I$ respectively, such that $\alpha\,|_{\mathbf{X}}$ is a $(\mathbf{X}, I)$-variable assignment and $\alpha(x) \in (\alpha(T))^I$ if $x : T$, where T is a $(\mathbf{F}, \mathbf{X})$-type term. The assignment α is extended to a partial function α^I mapping terms (possibly containing variables) to their interpretations.*

[2] Remark that the interpretation of *choose* is not restricted—it may be any interpretation realising a choice.

$\overline{\Sigma}$-substitutions are defined as usual as $(\mathbf{X} \cup \mathcal{X}, I)$-variable-assignments α in $\overline{\Sigma}$-term interpretations I, such that $\{x \in \mathbf{X} \cup \mathcal{X} \mid \alpha(x) \neq x\}$ is finite. The set $\{x \in \mathbf{X} \cup \mathcal{X} \mid \alpha(x) \neq x\}$ is then called the domain of α, written $\mathcal{D}om(\alpha)$. The set of $\overline{\Sigma}$-substitutions with domain included in $\mathcal{X}$ is written $SUBST_{\mathbf{\Sigma}}(\mathbf{X} \cup \mathcal{X})$.

Now, given a $\overline{\Sigma}$-interpretation I, we can define the truth value of a formula by universal quantification over all $(\mathbf{X} \cup \mathcal{X}, I)$-variable assignments.

Definition 3.8 *Let I be a $\overline{\Sigma}$-interpretation and α be a $(\mathbf{X} \cup \mathcal{X}, I)$-variable assignment.*

- *An existential formula $T \downarrow$ holds in I under α if $\alpha^I(T)$ is defined.*
- *An equality $S = T$ holds in I under α if $\alpha^I(S)$ and $\alpha^I(T)$ are both defined and are identical.*
- *A membership formula $S \in T$ holds in I under α if $\alpha^I(S)$ and $\alpha^I(T)$ are both defined and $\alpha^I(S)$ is an element of $\alpha^I(T)$.*
- *An implication $A_1 \wedge \ldots \wedge A_n \Rightarrow A$ holds in I under α if A holds in I under α whenever all of $A_1, \ldots, A_n$ hold in I under α. A negative implication $A_1 \wedge \ldots \wedge A_n \Rightarrow$ holds in I under α if $A_1, \ldots, A_n$ never hold together in I under α. A trivial implication $\Rightarrow A$ holds in I under α if A holds in I under α.*

A $(\overline{\Sigma}, \mathcal{X})$-Horn clause holds in I if it holds in I for all $(\mathbf{X} \cup \mathcal{X}, I)$-variable assignments α. I is a model of a presentation $\mathcal{P}$ if all Horn clauses in $\mathcal{P}$ hold in I.

Formula satisfaction is written $I \models \Phi$, or $\mathcal{P} \models \Phi$ in case all models of $\mathcal{P}$ satisfy the formula Φ. $\mathcal{P}$ is consistent if it has a model.

3.3 Proof Rules

From this section on, we restrict the possible sets of formulas, such that some atoms, trivially unsatisfiable due to type mismatches, are eliminated.

Definition 3.9 *A* standard $\overline{\Sigma}$-presentation *is a $\overline{\Sigma}$-presentation $\mathcal{P}$, where:*

- *All non-variable terms occurring in $\mathcal{P}$ belong to a type of kind TTYPE.*
- *If a constant c (other than $\mathbb{U}$) is declared to be of type T, then $c \in T$ is a fact in $\mathcal{P}$.*
- *choose does not occur in the conclusion of any clause in $\mathcal{P}$.*
- *All variables of type T with $T : TYPE$ in $\mathcal{P}$ occur only as the left-hand-side of a membership relation '$\in$'.*
- *All equalities $s = t$ satisfy that the sets of types of $\sigma(s)$ and $\sigma(t)$ are the same for every $\overline{\Sigma}$-substitution σ.*

- *All membership formulas $s \in t$ satisfy that $\sigma(t)$ is of type $\mathbb{T}(S \rightarrow\!\!\!\!\!+ T)$, $\mathbb{T}(S \leftrightarrow T)$ or $\mathbb{T}(\mathbb{P}(S \times T))$, if $\sigma(s)$ is of type $S \times T$, and of type $\mathbb{T}(\mathbb{P}(T))$ if $\sigma(s)$ is of any other type, for every $\overline{\Sigma}$-substitution σ.*

The proof rules shown in Table 1 are for a deduction relation $\vdash$ taking two arguments. The first argument is a standard $\overline{\Sigma}$-presentation. The second argument is a $\overline{\Sigma}$-formula Φ. These rules are divided into four groups: the upper one is for the embedding of typing into set theory, and uses the subtype inclusion defined in Section 3.1. Then there are rules handling function and relation graphs. Note that z is here intended to be of type $\mathbb{U} \rightarrow\!\!\!\!\!+ \mathbb{U}$, which is a subtype of $\mathbb{P}(\mathbb{U} \times \mathbb{U})$. These are followed by general set theoretic rules, which correspond to the axiom of choice and the axiom of (non-standard) extensionality. Last but not least, there are the logic inference rules; the meta-variables G, G' range over possibly-empty conjunctions of atomic formulae, and L, L' may be a single atomic formula or empty.

One of the inference rules (**SubstConform**) uses $\mathcal{P}$-conform $\overline{\Sigma}$-substitutions. These are defined as follows:

Definition 3.10 *Let $\sigma \in SUBST_{\mathbf{\Sigma}}(\mathbf{X} \cup \mathcal{X})$ and $\mathcal{P}$ be a $\overline{\Sigma}$-presentation. Then σ is* $\mathcal{P}$-conform *if for any variable x in $\mathcal{D}om(\sigma) \cap \mathcal{X}$, $\mathcal{P} \vdash \sigma(x) \downarrow$. The set of all $\mathcal{P}$-conform substitutions is denoted by $\mathcal{P}$-$SUBST_{\mathbf{\Sigma}}(\mathbf{X} \cup \mathcal{X})$.*

$\mathcal{P}$-conform substitutions have the nice property that they behave like $\overline{\Sigma}$-substitutions for composition with general variable assignments.

3.4 Results

We first address correctness of the deduction rules.

Theorem 3.11 *Let $\mathcal{P}$ be a standard $\overline{\Sigma}$-presentation. Then, all deduction rules in* **PL** *are sound, i.e. if $\mathcal{P} \vdash \Phi$ then $\mathcal{P} \models \Phi$.*

Proof: We have to check that each rule is sound in any $\overline{\Sigma}$-interpretation which is a model of $\mathcal{P}$.

- **SubType**: if $t \in S$ holds in a $\overline{\Sigma}$-interpretation I, for any assignment α, $\alpha^I(t) \in \alpha^I(S)$. Then if $S \leq T$, $\alpha^I(S) \subseteq \alpha^I(T)$ (since when both $t \in S$ and $T \downarrow$ hold, neither S nor T can involve $\mathbb{U}$ at all, hence either $S = T$ or $S = \mathbb{T}(T)$). So it is sound to deduce that $\alpha^I(t) \in \alpha^I(T)$ for any assignment α, thus $t \in T$ holds in the $\overline{\Sigma}$-interpretation I,
- **PairType1**, **PairType2**, **AppTyp**, **FApply**, **FGraph**, **RApply**, **RGraph**: these axioms are valid in any $\overline{\Sigma}$-interpretation by construction.
- **SubSet**, **Choice** are sound due to the interpretation of $\mathbb{P}$ as a partial power set function and of *choose* as a choice function (defined on all non-empty sets) in every $\overline{\Sigma}$-interpretation.

SubType:	$\dfrac{t \in S,\, T\downarrow}{t \in T}$ if $S \leq T$
PairType1:	$x \in X \wedge y \in Y \wedge (x,y)\downarrow \wedge (X \times Y)\downarrow \Rightarrow (x,y) \in (X \times Y)$
PairType2:	$(x,y) \in (X \times Y) \Rightarrow x \in X \wedge y \in Y$
AppType:	$x \in X \wedge z \in (X \mathrel{+\!\!\!\rightarrow} Y) \wedge z(x)\downarrow \Rightarrow z(x) \in Y$
FApply:	$x \in X \wedge y \in Y \wedge z \in (X \mathrel{+\!\!\!\rightarrow} Y) \wedge (x,y) \in z \Rightarrow z(x) = y$
FGraph:	$x \in X \wedge y \in Y \wedge z \in (X \mathrel{+\!\!\!\rightarrow} Y) \wedge z(x) = y \Rightarrow (x,y) \in z$
RApply:	$x \in X \wedge y \in Y \wedge z \in (X \leftrightarrow Y) \wedge (x,y) \in z \Rightarrow z(x,y)$
RGraph:	$x \in X \wedge y \in Y \wedge z \in (X \leftrightarrow Y) \wedge z(x,y) \Rightarrow (x,y) \in z$
SubSet:	$x \in y \wedge y \in \mathbb{P}(z) \Rightarrow x \in z$
Choice:	$\dfrac{s \in t}{choose(t) \in t}$ if $t : T$ for some $T : STYPE$
Ext:	$\dfrac{choose(x) \in y,\, choose(y) \in x}{x = y}$ if $x, y : T$ for some $T : STYPE$
WellDef:	$\dfrac{\Phi[t]}{t\downarrow}$ if $\Phi[t]$ *is a* $(\overline{\Sigma}, \mathcal{X})$-atom containing t
PartialReflex:	$\dfrac{t\downarrow}{t = t}$
Axioms:	$G \Rightarrow L$ if $(G \Rightarrow L) \in \mathcal{P}$ is a $(\overline{\Sigma}, \mathcal{X})$-clause
SubstConform:	$\dfrac{G \Rightarrow L}{\sigma(G) \Rightarrow \sigma(L)}$ if $\sigma \in \mathcal{P}\text{-}SUBST_{\mathbf{\Sigma}}(\mathbf{X} \cup \mathcal{X})$
Cut:	$\dfrac{G \wedge L' \Rightarrow L,\, G' \Rightarrow L'}{G \wedge G' \Rightarrow L}$
Paramodulation:	$\dfrac{G \Rightarrow L[s],\, G' \Rightarrow (s = t)}{G \wedge G' \Rightarrow L[t]}$

Table 1: **PL** – Deduction Rules for $\mathcal{P} \vdash \Phi$

- **Ext** is sound and expresses extensional equality on non-standard sets, due to the fact that *choose* is free in every $\overline{\Sigma}$-interpretation. Extensional equality of two sets can only be deduced when the premisses hold for all interpretations of the *choose* function, i.e. for any element in the sets. This amounts to check that both sets have the same elements.
- **WellDef** is sound since if an atom $\Phi[t]$ is valid in every model of $\mathcal{P}$, any term t in this atom should denote an element in the model. This is due to the fact that atoms only hold when their argument terms denote elements—in particular, our equality is existential.
- **PartialReflex** is the restriction of reflexivity to well-defined terms, so is sound.
- **Axioms** is trivially sound in every model of $\mathcal{P}$.
- **SubstConform, Cut, Paramodulation** are standard sound rules for Horn clause logic with equality.

□

The following lemma is needed in order to show that the initial model defined below actually is a model. It can be proved by simple composition of $\mathcal{P}$-conform substitutions with variable assignments:

Lemma 3.12 *Let $\overline{\Sigma}$ be a signature, $\mathcal{P}$ a presentation, I a $\overline{\Sigma}$-interpretation, G a conjunction of $(\overline{\Sigma}, \mathcal{X})$-atoms, $\sigma \in \mathcal{P}\text{-}SUBST_{\mathbf{\Sigma}}(\mathbf{X})$ and δ be a variable assignment for $\mathbf{X} \cup \mathcal{X}$ in I. Then $\sigma(G)$ holds in I under δ iff G holds in I under $\delta \circ \sigma$.*

Proof: Let us first prove that if $\sigma \in \mathcal{P}\text{-}SUBST_{\mathbf{\Sigma}}(\mathbf{X})$ and δ is a variable assignment for $\mathbf{X} \cup \mathcal{X}$ in I, $\delta \circ \sigma(x)$ is also a variable assignment of $\mathbf{X} \cup \mathcal{X}$ in I. This amounts to prove that for all $x \in \mathcal{X}$ with $x : T$, $\delta \circ \sigma(x) \in \delta \circ \sigma(T)^I$. If $x \notin \mathcal{D}om(\sigma)$, then this is implied by δ being a $(\mathbf{X} \cup \mathcal{X}, I)$-variable assignment. Otherwise, we know that $\delta(\sigma(x)) \in \mathbb{T}(U_I)$, since $\mathcal{P} \models \sigma(x) \downarrow$. Moreover $\sigma(x) \in \sigma(T)$ by definition of a substitution as a special variable assignment in a term interpretation. Hence, $\delta(\sigma(x)) \in \delta(\sigma(T))^I$.

Then let us denote $I, \delta \models \sigma(G)$ the statement $\sigma(G)$ holds in I under δ. Assume that G is a conjunction of atoms $p(\ldots, t, \ldots)$. Then

$$\begin{array}{rll} I, \delta \models \sigma(p(\ldots, t, \ldots)) & \text{iff} & I, \delta \models p(\ldots, \sigma(t), \ldots) \\ & \text{iff} & (\ldots, \delta(\sigma(t)), \ldots) \in p^I \\ & \text{iff} & I, \delta \circ \sigma \models p(\ldots, t, \ldots). \end{array}$$

□

The initial model constructed in the next definition interprets individuals (t of type $\mathbb{T}(\mathbb{I})$), pairs (t of type $\mathbb{T}(S \times T)$) and non-standard terms (t of type $T : TYPE$) as representative of their equivalence class (written $[t]$), whenever they exist. Sets t (of type $\mathbb{T}(\mathbb{P}(T))$) and graphs of functions and relations are interpreted such that they contain exactly those term interpretations $[u]$ with u being provably in t, also only if t exists. Remark that this includes the interpretation of terms built from type constructors. $\mathcal{V}ar(t)$ denotes the set of variables of t.

Definition 3.13 *Let $\mathcal{P}$ be a standard $\overline{\Sigma}$-interpretation and $V \subseteq \mathbf{X} \cup \mathcal{X}$. We define $\mathbf{I}_{\mathcal{P}}(V)$ to be the $\overline{\Sigma}$-interpretation I whose universe U_I is the set of all $[t]$ where t is a $(\overline{\Sigma}, V)$-term with $\mathcal{P} \vdash t \downarrow$, and where $[.]$ is defined as follows:*

- *$\forall t$ with $\mathcal{V}ar(t) \subseteq V$, such that $t : T$ where $T : RTYPE$ or $T : TYPE$, $[t]$ is a representative of $\{u \mid u : T, \mathcal{V}ar(u) \subseteq V$ and $\mathcal{P} \vdash u = t\}$ if $\mathcal{P} \vdash t \downarrow$;*
- *$\forall t$ with $\mathcal{V}ar(t) \subseteq V$, such that $t : T$ with $T : STYPE$, $[t] = \{[u] \mid u : U, \mathcal{V}ar(u) \subseteq V$ and $\mathcal{P} \vdash u \in t\} \cup \{[choose(t)]\}$ if $\mathcal{P} \vdash t \downarrow$, where U is S when T is $\mathbb{T}(\mathbb{P}(S))$, and $R \times S$ when T is $\mathbb{T}(R \mathrel{+\!\!\!\rightarrow} S)$ or $\mathbb{T}(R \leftrightarrow S)$.*

For all constants c, let $c^I = [c]$. Finally, for any $[t_1], [t_2] \in U_I$: let $([t_1], [t_2])^I = [(t_1, t_2)]$ if $\mathcal{P} \vdash (t_1, t_2) \downarrow$, otherwise undefined; and let $[t_1]([t_2])^I = [t_1(t_2)]$ if $\mathcal{P} \vdash t_1(t_2) \downarrow$, otherwise undefined.

One may check that this definition is a $\overline{\Sigma}$-interpretation in the sense of Definitions 3.3 and 3.4.

The difficulty in the initiality proof is hidden in the extensionality of sets. The following lemma is crucial for the interpretation of equality and membership. It decomposes into three steps: (1) Equality on individuals (type $\mathbb{T}(\mathbb{I})$), pairs (type $\mathbb{T}(S \times T)$) and non-standard elements (type $T : TYPE$) coincides with deductive equality. (2) The same holds for the membership atoms in standard presentations. (3) This is true for equality of sets represented by standard terms.

Lemma 3.14 *Consider $[.]$ as defined in Definition 3.13.*

1. *For all $s, t : T$ with $T : TYPE$ or $T : RTYPE$, $\mathcal{P} \vdash s = t$ iff $[s] = [t]$.*
2. *For all $s : S$ with $S : TYPE$ or $S : RTYPE$, and $t : T$ with $T : STYPE$, $\mathcal{P} \vdash s \in t$ iff $[s] \in [t]$.*
3. *And for all $s, t : S$ with $S : STYPE$, $\mathcal{P} \vdash s = t$ iff $[s] = [t]$.*

Proof: The left-to-right direction is trivial by definition and the reflexivity, paramodulation, well-definedness rules of our logic. From right-to-left, let us assume $s : S$ and $t : T$.

1. $[s]$ is a representative of $\{u \mid u : T, \mathcal{V}ar(u) \subseteq V$ *and* $\mathcal{P} \vdash u = s\}$ and $[t]$ is a representative of $\{v \mid v : T, \mathcal{V}ar(v) \subseteq V$ *and* $\mathcal{P} \vdash v = t\}$. If $[s] = [t]$, there exists w such that $\mathcal{P} \vdash w = s$ and $\mathcal{P} \vdash w = t$. So $\mathcal{P} \vdash s = t$.

2. Assume that $\mathcal{P} \vdash t \downarrow$ and $\mathcal{P} \vdash s \downarrow$. $[s]$ is a representative of $\{u \mid u : S, \mathcal{V}ar(u) \subseteq V \ and\ \mathcal{P} \vdash u = s\}$, T is of the form $\mathbb{T}(\mathbb{P}(S))$, $\mathbb{T}(R \twoheadrightarrow S)$, or $\mathbb{T}(R \leftrightarrow S)$, and $[t] = \{[w] \mid w : W,\ \mathcal{V}ar(w) \subseteq V \ and\ \mathcal{P} \vdash w \in t\}$. W is S when T is $\mathbb{T}(\mathbb{P}(S))$, and $R \times S$ in the other two cases.

   If $[s] \in [t]$, there exists $[w]$ such that $[s] = [w]$ and $\mathcal{P} \vdash w \in t$. Since $s : S$ and $w : W$ (where W cannot be of kind $STYPE$, since S is not, and $[s]$ and $[w]$ must therefore both be representative terms rather than actual sets) from (1) we get $\mathcal{P} \vdash s = w$. Hence, **Paramodulation** gives us $\mathcal{P} \vdash s \in t$.

3. If $[s] = [t]$, then $[choose(s)] \in [t]$, since $[choose(s)] \in [s]$ by **Choice**. Using (2), we know now that we can derive $choose(s) \in t$ from $\mathcal{P}$. Analogously, we can derive $choose(t) \in s$ and therefore by **Ext** also $s = t$.

□

Now, we can prove that our candidate for an initial model is actually a model in our logics.

Theorem 3.15 *Let $\mathcal{P}$ be a consistent standard $\overline{\Sigma}$-presentation and $V \subseteq \mathbf{X} \cup \mathcal{X}$. Then $\mathbf{I}_{\mathcal{P}}(V)$ is a model of $\mathcal{P}$.*

Proof: Let I stand for $\mathbf{I}_{\mathcal{P}}(V)$ in the following.

Lemma 3.14 guarantees that the relations $\in$ and $=$ are interpreted correctly. Remark that **Paramodulation** and **PartialReflex** guarantee that the relation $=$ is a congruence. The satisfaction of existential formulas '$t \downarrow$' follows from the definition of the interpretation: by construction of I, $[t]$ is defined if and only if $t \downarrow$.

In order to prove that I is a model of $\mathbf{P}$, we prove that all clauses in $\mathcal{P}$ are valid in I. Suppose $G \Rightarrow L$ is in $\mathcal{P}$ and δ is a $(I, \mathbf{X} \cup \mathcal{X})$-variable assignment, such that $I \models \delta(G)$. Let us define the $\overline{\Sigma}$-substitution δ', such that $[\delta'(x)] = \delta(x)$ for all $x \in \mathbf{X} \cup \mathcal{X}$, i.e. $I \models \delta'(G)$ by Lemma 3.12 (use δ' as σ and δ defined by $\delta(x) = [x]$ for all $x \in \mathcal{X}$). So $I \models \delta'(A)$ for any atom A in the conjuction of atoms G. By definition of I and Lemma 3.14 $\mathcal{P} \vdash \delta'(A)$ and so $\mathcal{P} \vdash \delta'(G)$. Then we can use the **SubstConform** rule on $G \Rightarrow L$, followed immediately by k times **Cut** (where k is the number of atoms in G), in order to get $\mathcal{P} \vdash \delta'(L)$. Hence, $I \models \delta(L)$ by definition of I. □

There remains to show completeness of deduction and initiality. Both proofs are in fact similar to those for G^n-logics [6].

Theorem 3.16 *Let $\mathcal{P}$ be a consistent standard $\overline{\Sigma}$-presentation and L a $(\overline{\Sigma}, \mathcal{X})$-atom. $\mathcal{P} \models L$ if and only if $\mathcal{P} \vdash L$.*

Proof: Since the rules in **PL** are correct we already know that if $\mathcal{P} \vdash L$, then $\mathcal{P} \models L$. Conversely if $\mathcal{P} \models L$, since $\mathbb{I}^{\mathbf{I}_{\mathcal{P}}(V)}$ is a model of $\mathcal{P}$, we know that under any assignment α, $\mathbb{I}^{\mathbf{I}_{\mathcal{P}}(V)}, \alpha \models L$. Applying this to the assignment δ defined by $\delta(x) = [x]$ for any x in V, we get that $\delta(L)$ is valid in $\mathbb{I}^{\mathbf{I}_{\mathcal{P}}(V)}$. $\delta(L)$ is of one of the following forms: $[s] = [t]$, $[s] \in [t]$, or $[s] \downarrow$. From $\mathbb{I}^{\mathbf{I}_{\mathcal{P}}(V)} \models [s] = [t]$, $\mathbb{I}^{\mathbf{I}_{\mathcal{P}}(V)} \models [s] \in [t]$, $\mathbb{I}^{\mathbf{I}_{\mathcal{P}}(V)} \models [s] \downarrow$, we get by definition of $\mathbb{I}^{\mathbf{I}_{\mathcal{P}}(V)}$, $\mathcal{P} \vdash s = t$, $\mathcal{P} \vdash s \in t$, $\mathcal{P} \vdash s \downarrow$. So $\mathcal{P} \vdash L$. □

Theorem 3.17 *Let $\mathcal{P}$ be a consistent standard $\overline{\Sigma}$-presentation. For all models J of $\mathcal{P}$, there is a unique $\overline{\Sigma}$-homomorphism $h : \mathbf{I}_{\mathcal{P}}(\varnothing) \to J$.*

Proof: The homomorphism is defined by $h([t]) = t^J$ for any $\overline{\Sigma}$-term t. Let us check that it satisfies Definition 3.6. Let I be $\mathbf{I}_{\mathcal{P}}(\varnothing)$, J be any $\overline{\Sigma}$-interpretation, and $h : I \to J$ be the function $h : U_I \to U_J$ defined by $h([t]) = t^J$. It satisfies:

- for all $(\mathbf{F}, \varnothing)$-terms T, $[t] \in [T]$ implies $\mathcal{P} \vdash t \in T$, so $t^J \in T^J$. Then $h([t]) = t^J \in h([T]) = T^J$;
- for all constants $c \in \mathcal{F} \cup \mathbf{F}$ other than $\mathbb{U}$, $h([c]) = c^J$;
- for all $(\mathbf{F}, \varnothing)$-terms T_1, T_2 of kind $TTYPE$, for all $[t_1] \in [T_1]$ and $[t_2] \in [T_2]$, $h(([t_1], [t_2])^I) = h([(t_1, t_2)]) = (t_1, t_2)^J = (t_1^J, t_2^J)^J = (h([t_1]), h([t_2]))^J$ whenever $([t_1], [t_2])$ exists;
- for all $(\mathbf{F}, \varnothing)$-terms T, T' of kind $TYPE$, for all $[t] \in \mathbb{T}(T)^I$ and for all $[f] \in \mathbb{T}(T \nrightarrow T')^I$, $h([f]([t])^I) = h([f(t)]) = f(t)^J = f^J(t^J)^J = h([f])(h([t]))^J$ whenever $[f]([t])$ exists.

Unicity of the homomorphism h from I to J is proved as usual by structural induction on t. □

3.5 Pragmatics

The following extension to the grammar given in Section 3.1 covers the entire specification language used in the illustrations in Section 2:

SPEC	::=	*NAME DECLS CLAUSE* | *NAME* [*VARS*] *DECLS CLAUSE*
DECLS	::=	*DECL* | *DECL*, *DECLS*
DECL	::=	*NAME* | *CONST* : *TERM*
CLAUSE	::=	... | *ATOMS* ⇒ *CLAUSE* | *CLAUSE* ∧ *CLAUSE*
		| *ATOMS* ⇔ *ATOMS* | *ATOMS* | ∀ *VARS* : *TYPE* • *CLAUSE*
CONST	::=	... | _ → _
VARS	::=	*VAR* | *VAR*, *VARS*
TYPE	::=	*TERM*

The main points to note in the transformation from the above into the kernel language defined in Section 3.1 are these: a name used in a declaration is replaced by the declarations to which it refers, the associated variables and clauses being incorporated too; set-valued terms used as types in declarations have to be replaced by their element types (cf. Z); the same holds for types of quantified variables in clauses; each constant declaration $c : T$ generates a clause $c \in T$; each quantified variable declaration $x : T$ generates an atom $x \in T$ in the premiss of the clause where x occurs; declarations of total functions $f : S \rightarrow T$ generate $f : S \nrightarrow T$ and a clause $s \in S \Rightarrow f(s) \in T$; and clauses have to be converted to Horn form.

4 Illustrations of Proofs

Compared to Z specifications, our framework has some restrictions: we do not allow arbitrary quantifications, nor explicit negation. However this is a deliberate decision, since restricting our presentations to Horn clauses with equalities and membership atoms gives three great advantages:

- The operational semantics of our presentations is given by conditional rewriting. Our specifications are actually programs executable with an interpretor of rewrite rules.
- We claim that we can provide an automatic tool for building a conditional term rewriting system equivalent to a given presentation using a saturation technique.
- We also think possible to obtain with the same technique, a refutationally complete procedure for proving a theorem in the initial model of a presentation. This proof procedure uses an ordering on terms and atoms to apply inference rules only on maximal literals in the clauses, which reduces the search space.

Using the saturation procedure or the refutational theorem prover we have in mind, requires to be in equational Horn clause logic. So the first step is to establish a correspondance between deduction with rules in **PL** written $\mathcal{P} \vdash \Phi$, and deduction in Horn clause logic written $\mathcal{P} \vdash_{HCL} \Phi$.

Let us replace **SubType**, **Choice** and **Ext** by Horn clause axioms and consider the set **HA** of Horn axioms described in Table 2.

Now deduction rules **WellDef**, **PartialReflex**, **Axioms**, **SubstConform**, **Cut** and **Paramodulation** are deduction rules for partial Horn deduction. A little more work must be done for eliminating **WellDef**, **PartialReflex**, that requires to transform atoms $t\downarrow$ into $t \in \mathbb{U}$ (see [6]). Eventually every predicate different from equality has to be turned into a boolean function. Let us call Θ these two transformations. Then one can prove as in [6] that for any ground atom L, $\mathcal{P} \vdash L$ if and only if $\Theta(\mathcal{P} \cup \mathbf{HA}) \vdash_{HCL} \Theta(L)$.

However, for a better readability, in the examples below, we do not perform the transformation Θ.

Let us now briefly recall the saturation procedure as described for instance in [2, 16]. The three main inference rules are superposition into conclusion, superposition into premisses and equality resolution. Subsumption by another clause and elimination of tautologies are also used to eliminate redundant clauses.

SubType-H:	$t \in S \wedge T \downarrow \wedge S \leq T \Rightarrow t \in T$
PairType1:	$x \in X \wedge y \in Y \wedge (x, y) \downarrow \wedge (X \times Y) \downarrow \Rightarrow (x, y) \in (X \times Y)$
PairType2:	$(x, y) \in (X \times Y) \Rightarrow x \in X \wedge y \in Y$
AppType:	$x \in X \wedge z \in (X \nrightarrow Y) \wedge z(x) \downarrow \Rightarrow z(x) \in Y$
FApply:	$x \in X \wedge y \in Y \wedge z \in (X \nrightarrow Y) \wedge (x, y) \in z \Rightarrow z(x) = y$
FGraph:	$x \in X \wedge y \in Y \wedge z \in (X \nrightarrow Y) \wedge z(x) = y \Rightarrow (x, y) \in z$
RApply:	$x \in X \wedge y \in Y \wedge z \in (X \leftrightarrow Y) \wedge (x, y) \in z \Rightarrow z(x, y)$
RGraph:	$x \in X \wedge y \in Y \wedge z \in (X \leftrightarrow Y) \wedge z(x, y) \Rightarrow (x, y) \in z$
SubSet:	$x \in y \wedge y \in \mathbb{P}(z) \Rightarrow x \in z$
Choice-H:	$s \in t \Rightarrow choose(t) \in t$
Ext-H:	$choose(x) \in y \wedge choose(y) \in x \Rightarrow x = y$

Table 2: **HA** –Horn axioms

An ordered strategy is used for reducing the search space by using only maximal terms and literals w.r.t. a given ordering for inference computation. A saturation process is a sequence of presentations $(P_0, P_1, \ldots)$, also called a derivation, where P_i is deduced from P_{i-1} by application of one inference rule. This derivation must be fair in the intuitive sense that no clause is forgotten in the process of generating consequences. P_0 is consistent if and only if the empty clause does not belong to any P_i. Moreover if P_∞ is the set of persisting clauses in this fair derivation and does not contain the empty clause, then one can construct from P_∞ a conditional term rewriting system which is terminating and confluent in the initial model of P_0. This indeed provides a way to compute in a finite and unambiguous way the normal form of any expression in P_0. The complete description of the process and its proof can be found in [6].

For understanding the proof tools proposed below, we first show the transformation of the initial Z-style specification *NaturalNumbers* given in Section 2 into a presentation in our logic, according to Section 3.5. Note that specifying *succ* to be a total function would amount to adding to the following presentation the clause: $x \in \mathbb{N} \Rightarrow succ(x) \in \mathbb{N}$.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NaturalNumbers}} \\
\hline
\mathbb{N} : \mathbb{T}(\mathbb{P}(\mathbb{I})) \\
0 : \mathbb{T}(\mathbb{I}) \\
succ : \mathbb{T}(\mathbb{I} \rightarrowtail\!\!\!\!\rightarrow \mathbb{I}) \\
\_ \leq \_, \_ < \_ : \mathbb{T}(\mathbb{I} \leftrightarrow \mathbb{I}) \\
\hline
\forall\, x, y : \mathbb{T}(\mathbb{I}) \bullet \\
\quad\quad 0 \in \mathbb{N} \\
\quad\quad succ \in \mathbb{N} \rightarrowtail\!\!\!\!\rightarrow \mathbb{N} \\
\quad\quad \leq \in \mathbb{N} \leftrightarrow \mathbb{N} \\
\quad\quad < \in \mathbb{N} \leftrightarrow \mathbb{N} \\
\quad\quad x \in \mathbb{N} \Rightarrow 0 \leq x \\
\quad\quad x \in \mathbb{N} \wedge y \in \mathbb{N} \wedge x \leq y \Rightarrow succ(x) \leq succ(y) \\
\quad\quad x \in \mathbb{N} \Rightarrow 0 < succ(x) \\
\quad\quad x \in \mathbb{N} \wedge y \in \mathbb{N} \wedge x < y \Rightarrow succ(x) < succ(y) \\
\hline
\end{array}
$$

Considering this last set of clauses, with an adequate ordering on terms and formulas, and an ordered strategy, there is no possible superposition into the conclusions, neither into premises, so the presentation is saturated, hence consistent. The ordering may be built with a lexicographic path ordering (see for instance [4]) from a precedence $\succ$ including $\leq, <, succ \succ \mathbb{N}$. To order formulas, we may ignore the membership and equality relations, map atoms and terms to multisets of sequences, and compare sequences with the multiset extension of the lexicographic extension of $\succ$. For instance to compare the two atomic formulas $x \leq y$ and $x \in \mathbb{N}$, we compare $\{(\leq, x, y)\}$ and $\{(\mathbb{N}, x)\}$. Since $\leq \succ \mathbb{N}$, we get $(\leq, x, y) \succ_{lex} (\mathbb{N}, x)$, $\{(\leq, x, y)\} \succ_{lex}^{mult} \{(\mathbb{N}, x)\}$, and thus $x \leq y$ greater than $x \in \mathbb{N}$. It may be worth emphasising that with this saturation technique, we can handle a limited form of negative assertion. Assume for example that we want to state in our presentation that $\forall\, x \in \mathbb{N} \bullet \neg(x = succ(x))$. The negative clause $(x \in \mathbb{N}) \wedge (x = succ(x)) \Rightarrow$ is added to the presentation and superpositions are performed. With an ordered strategy, the saturation process terminates and proves the consistency of the whole presentation. Here to compare the two atomic formulas $x = succ(x)$ and $x \in \mathbb{N}$, we compare $\{(x), (succ, x)\}$ and $\{(\mathbb{N}, x)\}$. Since $succ \succ \mathbb{N}$, we get $(succ, x) \succ_{lex} (\mathbb{N}, x)$, $\{(x), (succ, x)\} \succ_{lex}^{mult} \{(\mathbb{N}, x)\}$, and thus $x = succ(x)$ greater than $x = succ(x)$ and $x \in \mathbb{N}$.

Then we can prove by refutation $\exists\, x \in \mathbb{N} \bullet \neg(x = succ(0))$. In this case, the clause $\Rightarrow (x \in \mathbb{N}) \wedge (x = succ(0))$ is added, and superposition is performed with a renaming of $(x \in \mathbb{N}) \wedge (x = succ(x)) \Rightarrow$ and generates $\Rightarrow (0 \in \mathbb{N}) \wedge (0 \in \mathbb{N})$, then the empty clause.

Taking $\mathbb{N}$ for X in *GenericSequences*, the specification gets transformed into the following one:

$$
\begin{array}{l}
\hline
\textit{NaturalSequences} \\
\hline
\textit{NaturalNumbers} \\
seq : \mathbb{T}(\mathbb{P}(\mathbb{I}) \mathrel{+\!\!\!\rightarrow} \mathbb{P}(\mathbb{I})) \\
\langle\,\rangle : \mathbb{T}(\mathbb{P}(\mathbb{I})) \\
\langle\_\rangle : \mathbb{T}(\mathbb{I} \mathrel{+\!\!\!\rightarrow} \mathbb{P}(\mathbb{I})) \\
\_ \frown \_ : \mathbb{T}((\mathbb{P}(\mathbb{I}) \times \mathbb{P}(\mathbb{I})) \mathrel{+\!\!\!\rightarrow} \mathbb{P}(\mathbb{I})) \\
length : \mathbb{T}(\mathbb{P}(\mathbb{I}) \mathrel{+\!\!\!\rightarrow} \mathbb{I}) \\
\hline
\forall\, x : \mathbb{T}(\mathbb{I}) \bullet \forall\, s, t, u : \mathbb{T}(\mathbb{P}(\mathbb{I})) \bullet \\
\quad seq \in \mathbb{P}(\mathbb{N}) \mathrel{+\!\!\!\rightarrow} \mathbb{P}(\mathbb{I}) \\
\quad \langle\,\rangle \in seq(\mathbb{N}) \\
\quad \langle\_\rangle \in \mathbb{N} \mathrel{+\!\!\!\rightarrow} seq(\mathbb{N}) \\
\quad \_ \frown \_ \in seq(\mathbb{N}) \times seq(\mathbb{N}) \mathrel{+\!\!\!\rightarrow} seq(\mathbb{N}) \\
\quad length \in seq(\mathbb{N}) \mathrel{+\!\!\!\rightarrow} \mathbb{N} \\
\quad s \in seq(\mathbb{N}) \Rightarrow \langle\,\rangle \frown s = s \\
\quad s \in seq(\mathbb{N}) \Rightarrow s \frown \langle\,\rangle = s \\
\quad s \in seq(\mathbb{N}) \wedge t \in seq(\mathbb{N}) \wedge u \in seq(\mathbb{N}) \\
\qquad\qquad \Rightarrow s \frown (t \frown u) = (s \frown t) \frown u \\
\quad length(\langle\,\rangle) = 0 \\
\quad x \in \mathbb{N} \wedge s \in seq(\mathbb{N}) \Rightarrow length(\langle x \rangle \frown s) = succ(length(s)) \\
\hline
\end{array}
$$

Extending the previous precedence with $length \succ seq, 0, succ$, this presentation is proved consistent. With similar arguments as previously, one can prove that it remains consistent if we add the following statement:

$$\nexists x \in \mathbb{N}, s \in seq(\mathbb{N}) \bullet \quad length(s) = length(\langle x \rangle \frown s).$$

The technique extends to the higher-order example of *MappingSequences* in Section 2. The difficulty again is to compare the terms $mapseq(f)(s \frown t)$ and $mapseq(f)(s) \frown mapseq(f)(t)$ with a suitable ordering. This can be done also by mapping each term to a sequence and comparing them with a lexicographic path ordering. The sequence corresponding with the first term is, e.g., $(mapseq, f, (\frown, s, t))$. The sequence corresponding with the second term is $(\frown, (mapseq, f, s), (mapseq, f, t))$. Assuming $mapseq \succ \frown$ in the precedence, $mapseq(f)(s \frown t)$ is greater than $mapseq(f)(s) \frown mapseq(f)(t)$, and superposition can be applied only on the first term. Then it is not difficult to see that the whole presentation is saturated, thus consistent.

However, in order to prove a theorem such as $\forall f, s \bullet \ length(mapseq(f)(s)) = length(s)$, we need an inductive theorem prover. Induction is also necessary to prove the totality of functions declared as partial, and this is certainly a domain for further investigations. This discussion leads to the conclusion that rewriting techniques offer good possibilities for (semi-)automated theorem proving in our framework. There is surely some work to adapt existing theorem provers for our logic, but the changes seem to be reasonably simple.

5 Conclusion

The framework proposed here has been largely compared to the abstract data types approach and to Z specifications in the previous sections. But other works have strong connections too.

- First of all, this work is derived from R^n-/G^n-logics presented in [8]. The construction of R^n-/G^n-logics starts with Russell's ramified theory of types, to build a set-theoretic framework providing expressive typing, higher-order functions and initial models at the same time. The parameter n, which is a natural number, gives a bound on the nesting depth of the sets used in interpretations. The difference from Russell's ramified theory of types is the consideration of non-term-generated models. R^n-logics are axiomatisable by an order-sorted equational Horn logic with a membership predicate, and G^n-logics provide in addition partial functions. The framework proposed in this paper is a refinement of G^n–logics with an extended type system. Kinds, not existing in G^n-logics, are added, and operators $\times$, $\nrightarrow$ and $\leftrightarrow$ provide further refinements of the type structure. The deduction rules are quite similar to those given for G^n-logics.

- ETL [10] is in fact a fragment of R^n-logics [6]. An ETL presentation is a triple $[\![\Omega, V, E]\!]$, such that Ω is a set of function symbols, V is a set of unsorted variables and E is a set of Ω-Horn clauses using only equality "=" and the typing relation ":" as binary operators. The typing relation satisfies the paramodulation axiom and therefore it can be used as a new relation symbol in our logic. Remark that we cannot reuse "$\in$" for this purpose, since it has more properties than ":" in ETL. So it is possible to construct a presentation, such that an Ω-atom is true in our logic if and only if it holds in ETL.

- Power and unified algebras have been proposed and compared in [15] and also greatly influenced our framework. Concerning power algebras, the main difference is the absence of a predefined empty set, which corresponds to the bottom element in unified algebra, and of predefined singletons. Instead we can define empty sets and singletons in our logic. The other operators of power algebras: inclusion, intersection and union, can be axiomatized in the same way as in unified algebras [15]. Hence we can encode power algebras (apart from extensional equality of sets, of course) in our framework, and come quite close to unified algebras.

- More generally this work has similarities with higher-order functional langages. Some higher-order features are provided, since function graphs are specified as set constants, which can be passed to other functions as higher-order arguments. During the last years, a number of papers have dealt with the extension of first-order algebraic specifications to higher-order ones. Among them are [9, 17, 12, 14, 13, 11]. Our approach differs from [11] for instance in that our formal basis is set-theoretic rather than purely algebraic and provides a uniform treatment of typing and higher-order functions. Compared with λ-calculi, we have deliberately omitted the abstraction operator in order to minimize the functions in the initial model, which may be crucial for limiting the search space for automated theorem proving.

References

[1] J. R. Abrial. *B-Tool Reference Manual.* Edinburgh Portable Compiler, 1991.

[2] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, pages 462–476, 1992.

[3] J. Dawes. *The VDM-SL Reference Guide.* Pitman, 1991.

[4] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.

[5] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and initial semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, 1985.

[6] C. Hintermeier. *Déduction avec sortes ordonnées et égalités.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, Oct. 1995.

[7] C. Hintermeier, H. Kirchner, and P. D. Mosses. Combining algebraic and set theoretic specifications. In M. Haveraaen, O. Owe, and O-J. Dahl, editors, *Recent Trends in Data Type Specification", Proc. 11th Workshop on Specification of Abstract Data Types joint with the 9th general COMPASS workshop. Oslo, Norway, September 1995. Selected papers*, volume 1130 of *Lecture Notes in Computer Science*, pages 255–273. Springer-Verlag, 1996.

[8] C. Hintermeier, H. Kirchner, and P. D. Mosses. R^n- and G^n-logics. In G. Dowek, J. Heering, K. Meinke, and B. Möller, editors, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 1074 of *Lecture Notes in Computer Science*, pages 90–108. Springer-Verlag, 1996.

[9] T. S. E. Maibaum and C. J. Lucena. Higher order data types. *International Journal of Computer and Information Sciences*, 9:31–53, 1980.

[10] V. Manca, A. Salibra, and G. Scollo. Equational type logic. *Theoretical Computer Science*, 77(1-2):131–159, 1990.

[11] K. Meinke. Universal algebra in higher types. *Theoretical Computer Science*, 100:385–417, 1992.

[12] B. Möller. Algebraic specifications with high-order operators. In L. Meertens, editor, *Proceedings IFIP TC2 Working Conf. on Program Specification and Transformation*, pages 367–392. IFIP, Elsevier Science Publishers B. V. (North-Holland), 1987.

[13] B. Möller, A. Tarlecki, and M. Wirsing. Algebraic specification with built-in domain constructions. In M. Dauchet and M. Nivat, editors, *Proceedings of CAAP'88*, Lecture Notes in Computer Science, pages 132–148. Springer-Verlag, 1988.

[14] B. Möller, A. Tarlecki, and M. Wirsing. Algebraic specifications or reachable higher-order algebras. In D. Sannella and A. Tarlecki, editors, *Recent Trends in Data Type Specification*, volume 332 of *Lecture Notes in Computer Science*, pages 154–169. Springer-Verlag, 1988.

[15] P. D. Mosses. Unified algebras and institutions. In *Proceedings 4th IEEE Symposium on Logic in Computer Science, Pacific Grove*, pages 304–312, 1989.

[16] R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In D. Kapur, editor, *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, volume 607 of *Lecture Notes in Computer Science*, pages 477–491. Springer-Verlag, 1992.

[17] A. Poigné. Partial algebras, subsorting and dependent types. prerequisites of error handling in algebraic specifications. In *Proceedings of Workshop on Abstract Data Types*, volume 332 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[18] J. M. Spivey. *Understanding Z: a specification language and its formal semantics.* Cambridge University Press, 1988.

[19] A. N. Whitehead and B. Russell. *Principia Mathematica*, volume 1. Cambridge University Press, Cambridge, MA, 1925.

[20] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, chapter 13, pages 675–788. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1990.

Recent Publications in the BRICS Report Series

RS-96-52 Claus Hintermeier, Hélène Kirchner, and Peter D. Mosses. *Combining Algebraic and Set-Theoretic Specifications (Extended Version)*. December 1996. 26 pp. Appears in Haveraaen, Owe and Dahl, editors, *Recent Trends in Data Type Specification: 11th Workshop on Specification of Abstract Data Types, joint with 8th COMPASS Workshop*, Selected Papers, LNCS 1130, 1996, pages 255–274.

RS-96-51 Claus Hintermeier, Hélène Kirchner, and Peter D. Mosses. *R^n- and G^n-Logics*. December 1996. 19 pp. Appears in Gilles, Heering, Meinke and Möller, editors, *Higher-Order Algebra, Logic, and Term-Rewriting: 2nd International Workshop*, HOA '95 Proceedings, LNCS 1074, 1996, pages 90–108.

RS-96-50 Aleksandar Pekeč. *Hypergraph Optimization Problems: Why is the Objective Function Linear?* December 1996. 10 pp.

RS-96-49 Dan S. Andersen, Lars H. Pedersen, Hans Hüttel, and Josva Kleist. *Objects, Types and Modal Logics*. December 1996. 20 pp. To be presented at the *4th International Workshop on the Foundations of Object-Oriented*, FOOL4, 1997.

RS-96-48 Aleksandar Pekeč. *Scalings in Linear Programming: Necessary and Sufficient Conditions for Invariance*. December 1996. 28 pp.

RS-96-47 Aleksandar Pekeč. *Meaningful and Meaningless Solutions for Cooperative N-person Games*. December 1996. 28 pp.

RS-96-46 Alexander E. Andreev and Sergei Soloviev. *A Decision Algorithm for Linear Isomorphism of Types with Complexity $Cn(log^2(n))$*. November 1996. 16 pp.

RS-96-45 Ivan B. Damgård, Torben P. Pedersen, and Birgit Pfitzmann. *Statistical Secrecy and Multi-Bit Commitments*. November 1996. 30 pp. To appear in *IEEE Transactions on Information Theory*.