International Journal of Foundations of Computer Science Vol. 2 No. 1 (1991) 43–65

UNIFORM ALGEBRAIC SPECIFICATIONS OF FINITE SETS WITH EQUALITY

J.A. BERGSTRA*

Programming Research Group, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands and Applied Logic Group, University at Utrecht, Utrecht, The Netherlands

S. MAUW*

Programming Research Group, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

F. WIEDIJK**

Programming Research Group, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Received 15 May 1990
Revised 15 January 1991
Communicated by John Tucker

ABSTRACT

We study a variety of ways to specify a two sorted structure involving the Booleans B, with constants *true* and *false* and a finite set D with an equality function *eq* to the Booleans, under the assumption that there is a constant (name) for each element of D. The specifications are evaluated with respect to several properties, like textual length, efficiency in execution and the use of special features.

Keywords: formal description techniques, abstract data types, algebraic specifications.

1. Introduction

1.1. Finite Sets with Named Objects and Equality

The purpose of this paper is to investigate the difficulties that arise if one specifies finite sets. In many practical specifications one needs to introduce finite collections of

* Sponsored in part by ESPRIT project 432 (METEOR).
** Sponsored by an NWO research grant.

objects. Usually each of these objects will have some mnemonic name and a declaration of the objects is needed that declares all their names and states that all objects are different. Let K be a finite collection of names with cardinality n. Then $eqD(K, n)$ denotes the two sorted algebra with sort names B and D and domains $B = \{T, F\}$ and $D = K$ equipped with a function eq of type $D \times D \to B$ such that $eq(d, d') = T$ if and only if $d = d'$. This means that the equality function exactly resembles the equality of objects in the algebra. With $\Sigma(K, n)$ we will denote the signature of $eqD(K, n)$. A uniform specification is a family of specifications $S(K, n)$ for all natural numbers n from 1 onwards and for all sets of names K with n elements, such that for each n, $S(K, n)$ is a specification of $eqD(K, n)$.

The reason to consider uniform specifications is that one expects the user of a specification language to choose a uniform specification $S(K, n)$ of the family $eqD(K, n)$ first and thereafter, whenever an algebra $eqD(K, n)$ must be specified, to use an instance of that specification. It is assumed that quite often a complex specification will involve the description of several subalgebras of the form $eqD(K, n)$. Indeed the different uniform specifications that will be indicated in this paper differ so much in style that it is hard to imagine that one uses representatives of different uniform specifications within one larger specification at the same time. Therefore whoever faces the need to specify several algebras of the form $eqD(K, n)$ will have to choose a uniform specification in his/her favourite specification language.

The definition of a uniform specification works for a multitude of specification languages. For a language additional requirements may be added. For example if one works with algebraic specifications (see [6]) a specification $S(K, n)$ will take the form $(\Sigma(K, n), E(K, n))$ with $\Sigma(K, n)$ some signature and $E(K, n)$ a collection of (conditional) equations. In the case of uniform algebraic specifications it is additionally required that the signature $\Sigma(K, n)$ extends $\Sigma(eqD(K, n))$ by a fixed auxiliary signature Σ' (independent of K and n).

We will use set K as a parameter of the specifications. Clearly only the cardinality of K really matters because a renaming suffices to obtain a specification for $eqD(K', n)$ from $eqD(K, n)$. Nevertheless larger K's will need longer names and of course the length of a specification is influenced by that. Every scheme to name elements of sets with cardinality n will need names with length up to $log(n)$ at least. The use of mnemonic names may lead to much longer names than needed to name n different objects however. Therefore we use $L(K)$ to denote the length of the longest name in K and we assume no *a priori* relation between n and $L(K)$ though for large K's a naming scheme with $L(K) = O(log\ n)$ is necessary and plausible. We will express the complexity of specifications $S(K, n)$ in terms of $L(K)$ and n.

1.2. Special Features for the Uniform Specification of the Family eqD(K, n)*

Now it is at least plausible, and for usual specification languages obviously true that a specification of *eqD(K, n)* must mention each name in *K* at least once. Then a textual size of *S(K, n)* of order *O(L(K).n)* becomes unavoidable under the assumption that the average name length in *K* is proportional to *L(K)*. We will indicate that this complexity can be trivially achieved as soon as the specification language used offers specialized syntactic features for this purpose. Suppose that our specification language contains the following construct.

```
new sort with equality
   protecting boolean values in
      BOOLNAME
   true represented by
      TRUENAME
   false represented by
      FALSENAME
begin
   sort name
      SORTNAME
   name of equality function
      EQNAME
   names of different elements
      NAME1, NAME2, ...., NAMEn
end
```

Assume that the semantics of this specification is to introduce a sort *SORTNAME* with *n* constants *NAME1-NAMEn* that are interpreted as different objects exhausting the domain of *SORTNAME* as well as a function *EQNAME* with domain *SORTNAME* × *SORTNAME* and co-domain *BOOLNAME* such that *EQNAME(x, y) = TRUENAME* if and only if $x = y$.

In the above circumstances the specification of *eqD(K, n)* for $n = 5$ and $K = \{one, two, three, four, five\}$ takes the form:

```
new sort with equality
   protecting boolean values in
      B
   true represented by
      T
   false represented by
      F
begin
   sort name
      D
   name of equality function
      eq
   names of different elements
      one, two, three, four, five
end
```

* With a feature we indicate a possible option of a specification language (e.g. the ability to handle mix-fix notation and so on).

Evidently uniform specifications can be given with length $(L(K) + c1).n + c2$ provided a specification language is used that allows a primitive specification of the mentioned kind. In the specification above $c1 = 2$ and $c2$ is just below 200. The value $c1 = 2$ arises from the comma and space that separate consecutive constant declarations. The constant $c2$ is found by counting all characters not part of the names in K.

This is of course asymptotically best possible. It will be a conclusion of our paper that there is much to be said indeed for offering special syntax for the uniform description of structures $eqD(K, n)$ within a specification language, because absence of such features may lead to disappointing complications.

The problem of course is what to do if one employs a specification language that fails to offer special syntax for the definition of the $eqD(K, n)$. It is impossible to investigate this matter for all specification languages at once, therefore we will restrict our attention to algebraic specifications, preferably of the kind that lead to complete (= confluent and terminating) term rewriting systems because these lend themselves for automatic execution as well as for automatic proofs of consistency. Indeed if general first order logic, dynamic logic or some higher order logics are used different solutions to our specification problem may be found.

1.3. Uniform Algebraic Specifications of eqD(K, n)

So the problem is to find specifications that introduce a new data type D in such a way that in the preferred semantics of the specification language D will denote a finite set and eq will denote a function from $D \times D$ to the Booleans that will tell exactly when two elements of D are equal. From the point of view of initial algebra specifications the existence of eq is not an obvious requirement. If one takes a signature Σ that declares a sort D and on D four constants *one*, *two*, *three* and *four*, then the algebraic specification $(\Sigma, \emptyset)$ will have as its initial algebra a set with four different elements. So in this semantics there is no need to say that the different constants are unequal. The need to add negative information about equality arises if further data types must be built upon D, for instance finite subsets of D with insertion and deletion. Then for the specification of deletion one will need an equational expression of inequality over D in order to specify an *if-then-else* construction where the condition is in fact an equation over D. This observation leads to the conclusion that what is really needed in many cases is a specification of D together with a function $eq: D \times D \to B$.

Now we present a more formal description of the situation. The basic signature Σ_0 contains the two sorts B and D, the constants *true* and *false* of sort B and a function eq from D times D to B. In short notation:

$$\Sigma_0 = \{S{:}B, S{:}D, C{:}true{:} \to B, C{:}false{:} \to B, F{:}eq{:}D \# D \to B\}.$$

Here we used S to indicate the declaration of sorts, C for constants and F for functions.

A *standard* Σ_0*-algebra A* satisfies the following three conditions (the interpretation function is denoted by [.], # determines the number of elements).

- $\#[B] = 2$
- $[true] \neq [false]$
- *For all* $x,y \in [D]$ $[eq](x,x) = [true]$
  $[eq](x,y) = [false]$ *iff* $x \neq y$

This means that a standard Σ_0*-algebra* contains a standard model of the Booleans and that the equality function behaves nicely.

Now we extend this basic signature with a set of names. Let CH be the alphabet containing the characters that will be used to construct the names of the constants. The set CH^+ contains all non-empty strings over CH. For $K \subset CH^+$, $K \neq \varnothing$ and $n = \#K$, define the signature $\Sigma(K, n)$ by

$$\Sigma(K, n) = \Sigma_0 \cup \{C{:}k{:} \rightarrow D \mid k \in K\}$$

A *standard* $\Sigma(K, n)$*-algebra A* satisfies three conditions:

- $A/_{\Sigma_0}$ is a *standard* Σ_0*-algebra*
- A is minimal
- $\#[D] = n$

This standard algebra thus consists of a standard interpretation of the Booleans and a set which contains exactly one element for each name in K, together with an equality function which behaves as expected.

For given $K \subset CH^+$, $K \neq \varnothing$ and $n = \#K$ it is easy to prove that all standard $\Sigma(K, n)$-algebras are isomorphic. We denote this isomorphism class by $S\Sigma(K, n)$.

The collection of all these classes is denoted by $S\Sigma(\omega)$.

$$S\Sigma(\omega) = \{S\Sigma(K, n) \mid K \subset CH^+, K \neq \varnothing, \ n = \#K\}$$

Finally we define a *uniform initial algebra specification F* as a class

$$\{(\Gamma_F \cup \Sigma(K, n), E_F(K, n)) \mid K \subset CH^+, K \neq \varnothing, n \in \omega\},$$

where

Γ_F is a fixed signature (i.e. independent of K and n),
$E_F(K, n)$ is a set of equations over $\Gamma_F \cup \Sigma(K, n)$,

such that

for all $K \subset CH^+$ $I(\Gamma_F \cup \Sigma(K, n), E_F(K))\,|_{\Sigma(K, n)} \in S\Sigma(K, n)$.

So a uniform initial algebra specification consists of a specification for each K and n, whose initial algebra is a standard $\Sigma(K, n)$-algebra when restricted to $\Sigma(K, n)$. Uniformity is expressed by the requirement that apart from the names in K the signature is fixed. Of course we expect the specifications of a uniform specification to have the same mechanism somehow, but we have not attempted to make that aspect formal in the definition of a uniform specification.

For practical reasons we will not work with the isomorphism class $S\Sigma(K, n)$, but we will choose a representation, which we call $eqD(K,<,n)$, or for short $eqD(K, n)$.

Let $K \subset CH^+$, $K \neq \emptyset$ and $n = \#K$ and let $<$ be a total order on K. Suppose $K = \{k_1, ..., k_n\}$ such that $k_i < k_{i+1}$ for $1 \leq i < n$. Now $eqD(K,<,n)$ is the algebra consisting of sorts $B = \{T,F\}$ and $D = \{1, ..., n\}$, whith the following interpretation:

$$[true] = T$$
$$[false] = F$$
$$[k_i] = i \quad for\ 1 \leq i \leq n$$
$$For\ all\ x,y \in D \qquad [eq]\ (x,x) = t$$
$$[eq]\ (x,y) = f \qquad iff\ x \neq y$$

In the case where we are not interested in the actual names in K, but merely in the cardinality of K, we will use the algebra $eqD(n)$, which is defined by:

$$eqD(n) = eqD(K, <, n)/_{\Sigma_0}.$$

The difficulty that makes it reasonable to devote a paper to this issue is that the most obvious uniform specifications of $eqD(K, n)$ take an excessive amount of space (proportional to n^2) whereas more concise specifications are either quite unnatural and tricky or unsatisfactory from the point of view of computational efficiency (i.e. the number of steps needed to find normal forms for expressions of the form $eq(d, d')$).

1.4. Nonexistence of a Generic Solution

The first mechanism that comes to mind when finding a uniform specification of the algebras $eqD(K, n)$ is a parameterized data type $S(P)$ having a formal parameter P with signature $\Sigma(eqD(\emptyset, 0))$. We would like that if an initial algebra specification $(\Sigma(eqD(K, n)), \emptyset)$ of a domain $D(K, n)$ with $n = \#(K)$ elements, named by the different names in K, is substituted for P, $S((\Sigma(eqD(K, n)), \emptyset))$ turns out to be a specification of an enrichment of $D(K, n)$ to $eqD(K, n)$.

However, one easily observes that such a parameterized data type specification S cannot exist. Indeed assume that S is specified by (Σ', E'), then for two names a and b not occurring in $\Sigma', E' \vdash eq(a, b) = F$. But equational logic is insensitive to the difference between a variable and a fresh name. So we can transform the proof of $eq(a, b) = F$ into a proof $E' \vdash eq(c, c) = F$ for any name c, by substituting c for both a and b. But on the other hand $E' \vdash eq(c, c) = T$. It follows that $E' \vdash T = F$, which leads to a contradiction.

It follows that the parameter n as well as the names in K must play a very explicit role in the construction of a uniform specification.

1.5. The Specification Language ASF

All specifications in this paper are written in the ASF language [2]. This language allows for modular specification by means of export, import and parameterization. Some

features of the language that are used are overloading of function symbols and infix operators.

2. Classifications of Algebraic Specifications of $eqD(K,n)$

2.1. A Listing of Possible Virtues

Let $S(K, n) = (\Sigma(K, n), E(K, n))$ be a uniform algebraic specification of the structures $eqD(K, n)$. This implies that $\Sigma(EqD(K, n)) \square I(S(K, n)) = eqD(K, n)$. Moreover we will read the equations of $E(K, n)$ from left to right as rewrite rules. Now the following virtues are worthwhile to achieve:

2.1.1. Conciseness

This is expressed in terms of the size of the specification, i.e. the best upperbound to the textual length of $S(K, n)$ as a function of n and $L(K)$ the length of the longest name in K. In all cases $L(K)$ only contributes a linear factor and the interesting part is the way in which n occurs in the size. As said in Sec. 1.2 it is not conceivable to do better in size than $L(K).n$ because each name must be mentioned at least once and all names may happen to have length $L(K)$.

2.1.2. Bounded number of equations

It is nice if the number of equations of a uniform specification does not depend on its parameters. For instance in [4] it was shown that all minimal finite algebras have an equational initial algebra specification involving three auxiliary functions, no auxiliary sorts and four equations. So this works for our problem as well. The specifications provided in that proof have no virtue as TRS's however.

If the number of equations is not uniformly bounded then at least one may require this number $\#(E(K, n))$ to grow as slow as possible. Notice that $\#(E(K, n))$ will not depend on K or $L(K)$ but only on n.

2.1.3. Absence of auxiliary sorts

From a theoretical point of view a specification is more attractive if it avoids the use of auxiliary sorts. For more interesting specifications the use of auxiliary functions seems to be unavoidable.

In [3] it was shown that every minimal computable algebra can be equationally specified with a complete term rewriting system using auxiliary functions only. Therefore this goal can always be met.

For infinite algebras in addition [3] provides uniform specifications with a number of equations proportional to the size of the signature of the algebra.

2.1.4. Completeness

(That is confluency and termination of the specifications viewed as a TRS.) Here it is understood that on B the normal forms must be T and F. This is a rather fundamental

requirement from a practical point of view. If this requirement is omitted one can obtain quite concise specifications (see Sec. 2.1.2 above.)

2.1.5. Equations rather than conditional equations.

Obviously it is nicer to use only equations

2.1.6. Standard normal forms

We will say that a specification has standard normal forms if for both B and D the normal forms are exactly the constants.

2.1.7. Fast termination

If the specifications are complete term rewriting systems one hopes that closed expressions e of the form $e = eq(t, t')$ will reduce to normal form with short reduction sequences. Efficient reduction is present if the number of steps needed is proportional to the length of e. Inefficient reduction is present if the length of reductions is proportional to n. The latter is the case with specification $S5$. For expressions $e = eq(a, b)$ with $a, b \in K$ this inefficient reduction will involve an exponential blow up of the time needed for reduction as a function of $length(e)$, provided the name space K uses reasonably short names (i.e. $L(K) = O(\log n)$).

In practice not only the length of the reduction path determines the efficiency of rewriting. The number of equations is also significant, since finding a left hand side of a rewriting rule that matches a given term could take time of order $\log(\#(E(K, n)))$.

On basis of the possible nice properties of uniform specifications for the family $eqD(K, n)$ one can find a simple classification. For instance our interest is in specifications that yield complete term rewriting systems.

2.2. *Uniform Algebraic Specification of eqD(K, n) Without Auxiliary Sorts and Functions*

The simplest uniform specifications involve no auxiliary sorts and functions at all. The foremost example is: $S1(K, n) = (\Sigma(eqD(K, n)), E1(K, n))$ where

$$E1(K, n) = \{eq(a, a) = T \mid a \in K\} \cup \{eq(a, b) = F \mid a \in K \,\&\, b \in K \,\&\, a \neq b\}.$$

The size of this specification is $(L(K) + c).n^2$. The specification provides a confluent TRS and expressions reduce in a single step to their normal form. So the performance of this specification is optimal from the point of view of efficient reduction but deplorable in terms of textual length. If $\#(K) = 2$, i.e. in the case of the Booleans $S(K, 2)$ is just the familiar truth table for the double implication and no much better ways of defining such functions are known. In the case of $n = 3$ this mechanism is acceptable as well.

A slightly shorter family of specifications is obtained as follows:

$S2(K, n) = (\Sigma(eqD(K, n)), E2(K, n))$, where
$E2(K, n) = \{eq(x, x) = T\} \cup \{eq(a, b) = F \mid a \in K \,\&\, b \in K \,\&\, a \neq b\}.$

The price paid for this optimization is that the TRS is not regular any more (because it is not left linear).

Remark. Using the priority rewrite rules of [1] a significant simplification is possible. $S3(K, n) = (\Sigma(eqD(K, n)), E3)$. Here $E3$ contains only two rules:

(1) $eq(x, x) = T$
(2) $eq(x, y) = F$

and (1) takes priority over (2). The problem with this kind of specification is that equality in a specification using priority rewrite rules is in general not decidable, so this specification will in general not be executable. (However, when we restrict ourselves to semi-complete priority rewrite systems, adding these two rules leaves the system semi-complete and still executable). The use of it is that it allows rewrite rule techniques to be used for more high level specifications, and this one is a convincing example.

3. Specification with Auxiliary Sort and Functions

3.1. Enumerated Type

The simplest specification of $eqD(K, n)$ using auxiliary material introduces the natural numbers with 0, successor and equality mapping. On the Booleans and Naturals we have defined some extra functions which will be needed subsequently in this paper.

```
module BOOL
begin
   exports
   begin
      sorts
         B
      functions
         T: → B
         F: → B
         eq: B × B → B
         _.and._: B × B → B
         _.or._ : B × B → B
   end
   equations
      eq(F, F)  = T
      eq(F, T)  = F
      eq(T, F)  = F
      eq(T, T)  = T
      F .and. F = F
      F .and. T = F
      T .and. F = F
      T .and. T = T
      F .or. F  = F
      F .or. T  = T
      T .or. F  = T
      T .or. T  = T
end BOOL
```

```
module NATL
begin
   exports
   begin
      sorts
         N
      functions
         0: → N
         s: N → N
         eq: N × N → B
         gteq: N × N → B --greater or equal
   end
   imports
      BOOL
   variables
      x, y: → N
   equations
      eq(0, s(x))       = F
      eq(s(x), 0)       = F
      eq(0, 0)          = T
      eq(s(x), s(y))    = eq(x, y)
      gteq(0, s(x))     = F
      gteq(x, 0)        = T
      gteq(s(x), s(y)) = gteq(x, y)
end NATL

module D
begin
   exports
   begin
      sorts
         D
      functions
         one  : → D
         two  : → D
         three: → D
         four : → D
         five : → D
   end
end D
```

Using these modules the structures *eqD(K, n)* are specified below in module *S4* by mapping each constant for D to a number in N and deriving *eq* on D from *eq* on N. We present the specification for $K = \{one, two, three, four, five\}$ and the generalization of this uniform specification *S4(K, n)* to all K and n is obvious. The size of these specifications is bounded by $(L(K) + {}^1/_2\, n + c1).n + c2$ where $c1$ counts the syntactic overhead per equation and $c2$ counts the overhead introduced by the imports and declarations. Obviously this is not satisfactory. Moreover the efficiency of reduction is deplorable as well because *eq(a, b)* may take up to n steps whereas the length of a and b is bounded by $L(K)$ which in

turn will often be of the order *O(log n)*. It follows that this specification is hardly better than *S1(K, n)*.

```
module S4
begin
   exports
   begin
      functions
         f : D      → N
         eq: D × D → B
   end
   imports
      BOOL, NATL, D
   variables
      x, y: → D
   equations
      f(one)    = 0
      f(two)    = s(0)
      f(three)  = s(s(0))
      f(four)   = s(s(s(0)))
      f(five)   = s(s(s(s(0))))
      eq(x, y)  = eq(f(x), f(y))
end S4
```

3.2. Enumeration by Chain

Some improvement of *S4(K, n)* is obtained if the definition of the embedding function f is made by explicit recursion along the elements of K. This approach is comparable with the approach taken in [8].

```
module S5
begin
   exports
   begin
      functions
         f : D      → N
         eq: D × D → B
   end
   imports
      BOOL, NATL, D
   variables
      x, y: → D
   equations
      f(one)    = 0
      f(two)    = s(f(one))
      f(three)  = s(f(two))
      f(four)   = s(f(three))
      f(five)   = s(f(four))
      eq(x, y)  = eq(f(x), f(y))
end S5
```

In this case the size of *S5* is bound by $(2.L(K) + c1).n + c2$ (The constants $c1$ and $c2$ vary from specification to specification). So this is much better by removing the quadratic dependence on n. Nevertheless normalization will now be extremely slow. Indeed if a is the last name of the enumeration the reduction of $eq(a, a)$ to T will take $3.n$ steps, which is about the worst one can imagine.

3.3. Enumeration to Binary Number Representation

A further improvement goes back to the idea of *S4* but replaces the specification of *NATL* by one that is based on a binary number representation. Naturals in binary notation are formed by sequences of bits, preceded by the constant *bin*, which represents the starting bit *1* of a number. We use the Boolean values to represent the bits: *T* stands for *1* and *F* stands for *0*. Thus the number *19*, which in binary notation is *10011*, is represented by *bin^F^F^T^T*.

```
module NATB
begin
   exports
   begin
      sorts
         N
      functions
         bin:        → N
         _^_: N × B → N
         eq : N × N → B
         gt : N × N → B --greater then
   end
   imports
      BOOL
   variables
      x, y: → N
      b, c: → B
   equations
      eq(bin, bin) = T
      eq(bin, y^c) = F
      eq(x^b, bin) = F
      eq(x^b, y^c) = eq(x, y) .and. eq(b, c)
      gt(bin, y)   = F
      gt(x^b, bin) = T
      gt(x^b, y^c) = gt(x, y) .or.
                     (eq(x, y) .and. eq(b, T) .and. eq(c, F))
end NATB
```

```
module S6
begin
   exports
   begin
      functions
         f: D → N
         eq: D × D → B
   end
   imports
      BOOL, NATB, D
   variables
      x, y: → D
   equations
      f(one)    = bin
      f(two)    = bin^F
      f(three)  = bin^T
      f(four)   = bin^F^F
      f(five)   = bin^F^T
      eq(x, y)  = eq(f(x), f(y))
end S6
```

The size of *S6* is bounded by $(L(K) + 2.\log n + c1).n + c2$. In the used format $c1 \leq 10$. This is definitely better than *S4* because $\log n << 1/2\ n$. Moreover normalization goes reasonably fast (order $O(\log(n))$). On top of this the size of *S6* is less than that of *S5* as soon as $2.\log n < L(K)$, which is very likely to be the case.

Note that the base of the logarithm is 2, since we used a binary representation of the naturals. Any higher base, up to the size of the character set, can be obtained, but this would increase the constant *c2* considerably.

This specification of $eqD(K, n)$ is probably the best one for practical purposes, although its level of abstraction is somewhat disappointing.

4. Uniform Specifications of $eqD(K, n)$ with Auxiliary Functions Only

In this section we will provide a specification that avoids auxiliary sorts. Its virtue moreover is that the constants of *D* are normal forms of the corresponding complete TRS. If that condition is dropped even more concise and efficiently reducing specifications without hidden sorts can be found. We provide the specification with $n = 5$. The general case is obvious.

```
module S7
begin
   exports
   begin
      functions
         eq: D × D → B
   end
   imports
      BOOL, D
   functions
      s: D → D
      gteq: D × D → B        -- greater then or equal
   variables
      x, y: → D
   equations
      s(one)   = two
      s(two)   = three
      s(three) = four
      s(four)  = five
      s(five)  = five

      gteq(one, five)   = F
      gteq(two, five)   = F
      gteq(three, five) = F
      gteq(four, five)  = F
      gteq(five, five)  = T

      gteq(x, one)   = gteq(s(x), two)
      gteq(x, two)   = gteq(s(x), three)
      gteq(x, three) = gteq(s(x), four)
      gteq(x, four)  = gteq(s(x), five)

      eq(x,y) = gteq(x,y) .and. gteq(y,x)
end S7
```

The number of equations for *S7* is *3.n*. The size of *S7(K, n)* is a rather awkward expression: *(6.L(K) + c1).n - 2.L(K) + c2* for appropriate *c1* and *c2*. Here, when working with the character set of the above specification $c1 \leq 50$. It follows that *S7* is never better than *S6*, in terms of size. In terms of reduction efficiency it is quite slow as well.

5. Uniform Specifications of *eqD(n)*

5.1. The Family of Structures eqD(n)

Although for practical purposes specifications of *eqD(K, n)* seem to be more important there is a mathematical reason to investigate specifications of finite sets with equality function without the constraint that every element of the set is the interpretation of a constant. Refer to Sec. 1.3 for a definition of *eqD(n)*. The motivation for studying uniform specifications of *eqD(n)* is not an independent one as for the case of *eqD(K, n)*. We consider this matter because it is technically intriguing and moreover we will find an interesting application on the uniform specifications for *eqD(K, n)* below.

5.2. A Parameterized Specification of eqD(n)

In this specification we use the auxiliary sort N and generate the sort D by a mapping f from N to D. A conditional axiom ensures that D collapses to the right size. The equation $eq(f(x), f(y)) = eq(x, y)$ then translates determination of eq on D to determination of eq on N.

```
module S8
begin
   parameters max
   begin
      functions
         max: → N     -- number of elements
   end max
   exports
   begin
      sorts D
      functions
         f: N → D     -- projection
         eq: D × D → B
   end
   imports
      BOOL, NATB
   variables
      x, y: → N
   equations
      f(x) = f(max)
         when
         gt(x, max) = T
      eq(f(x), f(y)) = eq(x, y)
         when
         gt(x, max) = F,
         gt(y, max) = F
end S8

module 16
begin
   exports
   begin
      functions
         16: → N
   end
   imports NATB
   equations
       16 = bin^F^F^F^F
end 16

module D16
begin
   imports S8
   { renamed by [D → D16]
    max bound by [max → 16] to 16 }
end D16
```

5.3. A Parameterized Specification of eqD(n) Needs Conditional Equations

In Sec. 1.4 we found that a parameterized data type will not provide a uniform specification of the family *eqD(K, n)*. The proof rests on inspections of the constant names in K. In the case of *eqD(n)* these constant names are absent, and a parameterized solution as in Sec. 5.2 could be found. Nevertheless in this way one will not obtain satisfactory term rewriting systems because it is not possible to avoid conditional equations. This can be shown as follows.

Suppose a parameterized specification of *eqD(n)* has the form of a module *M(NAT(c))* that imports a module *NAT(c)* which contains natural numbers with appropriate constructors and a constant c. The different specifications for *NAT(c)* may only vary in their single rewrite rule for c. Let us assume that all rules of the form $c = S^n(0)$ may occur. We write *NAT(c, n)* for the specification of *NAT(c)* that contains the rule $c = S^n(0)$. Assume that for each $n \geq 1$, *M(NAT(c, n))* is an equational initial algebra specification of *eqD(n)* which moreover is a complete term rewriting system. For each n there will be exactly n closed normal forms of sort D w.r.t. the TRS *M(NAT(c, n))*. But the only difference between these term rewriting systems is in the right-hand side of a rule. Therefore all must have exactly the same normal forms, which leads to a contradiction.

It follows that the standard parameterization mechanism as available in most languages for algebraic specification (such as Act-one [5], ASF [2], and OBJ2 [7]) cannot simulate the abstraction of parameter n from a uniform specification by means of parameterization.

5.4. Stronger Parameterization Mechanisms

The question whether a parameterization mechanism that is more powerful than the standard one suffices to construct a parameterized specification of *eqD(n)*, will be answered in this section. The first generalization is that we not only allow parameters to be bound to constants, but also to (closed) terms. This will turn out not to be sufficiently powerful.

5.4.1. Binding parameters to closed terms is too weak

We prove that there does not exist a parameterized equational specification of *eqD(n)*, even when we can bind the parameter to closed terms. We want such a specification to be a semi-complete rewriting system, which does not affect the normal forms of the natural numbers (i.e. a persistent extension of the naturals).

Suppose we have such a specification (Σ, E_k) with parameter k, of sort N. For the moment think of the naturals being specified as in the module *NATL*, with two constructor functions 0 and s. From this we are going to derive a contradiction.

We can split up the equations of E_k into two parts: the equations that contain the parameter k as part of the left-hand side (called T_k) and those that do not (called T).

Furthermore let d be defined as the maximal depth* of all left-hand sides in E, where we consider k as a constant term of depth *1*.

Let n be some natural number $> d$. Because we demand all specifications to be a semi-complete rewriting system each element in any sort corresponds to a normal form. Therefore, the finite sort in E_{n+1} has $n+1$ normal forms, whereas E_n has only n. This implies that there exists some closed term t of the finite sort which is a normal form of E_{n+1}, but is reducible in E_n.

We can prove that t is not only reducible in E_n, but already in T_n, because t is a normal form of E_{n+1} and therefore does not rewrite by rules from T. Since t reduces by a rule from T_n, (that contains a term $s^n(0)$ at the left hand side), we find that t has a subterm of the form $s^n(0)$.

Now define the term t_m as the term t, where all occurrences of $s^n(0)$ are replaced by $s^{n+m}(0)$. This construction forces all occurrences of $s^{n+1}(0)$ in t_m to be contained in an occurrence of the form $s^{n+m}(0)$.

Because t has a subterm of the form $s^n(0)$, all t_m are different. We now choose $m > d$, with the property that t_m is not a normal form of E_{n+1}. We can do this because E_{n+1} has only finitely many normal forms. Because t_m is reducible in E_{n+1} we can also find some subterm t'_m of t_m that is a redex with respect to E_{n+1}.

There are two options: (i) either t'_m is a redex of a rule from T, (ii) or t'_m is a redex of a rule in T_{n+1}.

(i) Suppose t'_m is a redex of a rule from T, then also the corresponding** subterm t' of t reduces in T, because both terms are equal up to depth d. Because t' reduces in E_{n+1} we can conclude that t also reduces in E_{n+1}, which leads to a contradiction.

(ii) Suppose t'_m is a redex of a rule from T_{n+1}, then t'_m has a subterm $t''_m = s^{n+1}(0)$ that corresponds to the $s^{n+1}(0)$ in the rule from T_{n+1}. Because of the construction of t_m, t''_m is contained in a greater subterm of t_m of the form $s^{n+m}(0)$. But all rules in T_{n+1} have maximal depth $d+n+1$, which is $\leq n+m$. Therefore the complete redex t'_m has to be contained in that greater subterm $s^{n+m}(0)$. So t'_m is of the form $s^i(0)$. We find that $s^i(0)$ is reducible which leads to a contradiction (since we postulated persistence of the naturals).

Remark. This proof can be generalized to the case where the natural numbers are represented by some other free sort.

5.4.2. Binding parameters to (open) terms suffices

If we allow an even more general parameterization mechanism, a parameterized specification of *eqD(n)* can be given. We will allow the possibility of having a function as

* The depth of a term is defined by:
depth(c) = 1 for constants, *depth(v) = 0* for variables and $depth(f(t_1, ..., t_n)) = 1+max(depth(t_i))$.

** We will not give a formal definition of this notion. Since t'_m is not of the form $s^i(0)$ this can easily be done.

a parameter, and to bind this function to any (open) term of the right sort. In specification *S9* the number of elements in *D* will be *max(0)+1*.

```
module S9
begin
   parameters max
   begin
      functions
         max: N → N
   end max
   exports
   begin
      sorts
         D
      functions
         f: N → D
         eq: D × D → B
   end
   imports
      BOOL, NATL
   variables
      x, y: → N
   equations
      f(s(max(x))) = f(max(x))
      eq(f(x),f(y)) = eq(x,y) .or.
                      (gteq(x,max(0)) .and. gteq(y,max(0)))
end S9

module D5
begin
   imports S9
      {max bound by
         variables x: → N
         [max(x) → s(s(s(s(s(x)))))] to NATL}

end D5
```

5.5. A Uniform Specification of eqD(n) with a Bounded Number of Equations

We will use the modules *BOOL* and *NATL* from Sec. 3.1. We consider the special case $n = 5$.

```
module NATL-ext
begin
   exports
   begin
      functions
         s5: N → N
   end
   imports
      BOOL, NATL
   functions
      min: N × N → N
```

```
    variables
        x, y: → N
    equations
        min(0,x) = 0
        min(x,0) = 0
        min(s(x),s(y)) = s(min(x,y))
        s5(x) = min(s(s(s(s(s(0))))), s(x))
end NATL-ext

module S10
begin
    imports
        BOOL, NATL-ext
    sorts
        D
    functions
        0D: → D
        s: D → D
        eq: D × D → B
        f: D → N
    variables
        x, y: → D
    equations
        eq(x, y) = eq(f(x), f(y))
        f(0D) = 0
        f(s(x)) = s5(f(x))
        s(s(s(s(s(0D))))) = s(s(s(s(0D))))
end S10
```

The uniform specification *S10(n)* provides complete term rewriting systems, but these show slow normalization. The size of the specification is linear in n.

5.6. A Concise Uniform Specification of eqD(n) Using One Auxiliary Constant and a Binary Auxiliary Function

We present the specification in detail in the case $n = 19$. From that the reader can easily find the general case, and by taking a specific instance tedious general notation is avoided. We denote the domain with *D19*. The idea of the specification is as follows: on *D* one introduces a constant *bin* representing the number *1* in a binary notation. Then we have a binary function $\_\hat{}\_ : D19 \times B \to D19$. This function constructs new numbers in the following way: $bin = 1$, $bin\hat{}F = 2$, $bin\hat{}T = 3$, $bin\hat{}F\hat{}F = 4$, $bin\hat{}F\hat{}T = 5$, ... , $bin\hat{}F\hat{}F\hat{}F\hat{}F = 16$, ... , $bin\hat{}F\hat{}F\hat{}T\hat{}T = 19$. In order to avoid generating more than *19* objects, two types of identifications are used:

(i) all expressions $bin\hat{}F\hat{}T\hat{}F\hat{}F$ up to $bin\hat{}T\hat{}T\hat{}T\hat{}T$ are identified with *bin* (i.e. *20-31* are collapsed into *1*)

(ii) all expressions of the form $f\hat{}x\hat{}y\hat{}z\hat{}u\hat{}v$ are identified with $f\hat{}v$ (i.e. *32* collapses to *2* and so on.

These identifications can be imposed by means of at most *log n* equations of length *log n*. Then the equality function must be specified. For all normal forms of equal length this is a matter of comparing the corresponding bits. For all normal forms of different length it must be said that they are different. Because there are *O(log n)* lengths in use the latter leads to $O(\log^2 n)$ equations. These equations have length *O(log n) (log log n)*. The *log n* is due to the length of the expressions in terms of operators and variables. The factor *log log n* stems from the fact that the various variables must get a name and *O(log log n)* names are needed for this mechanism. Altogether these specifications have length order $(\log n)^3 (\log \log n)$.

```
module D19
begin
   exports
   begin
      sorts D19
      functions
         bin: → D19
         _ ^ _: D19 × B → D19
   end
   imports BOOL
   variables
      x00, x01, x10, x11, x, b: → B
   equations
      bin ^ F ^ T ^ x01 ^ x00 = bin
      bin ^ T ^ x10 ^ x01 ^ x00 = bin
      bin ^ x11 ^ x10 ^ x01 ^ x00 ^ b = bin ^ b
end D19

module S11
begin
   exports
   begin
      functions
         eq: D19 × D19 → B
   end
   imports
      BOOL, D19
   variables
      x00, x01, x10, y00, y01, y10: → B
```

```
equations
    eq(bin, bin) =
        T
    eq(bin ^ x00, bin ^ y00) =
        eq(x00, y00)
    eq(bin ^ x01 ^ x00, bin ^ y01 ^ y00) =
        eq(x01, y01) .and. eq(x00, y00)
    eq(bin ^ x10 ^ x01 ^ x00, bin ^ y10 ^ y01 ^ y00) =
        eq(x10, y10) .and. eq(x01, y01) .and. eq(x00, y00)
    eq(bin ^ F ^ F ^ x01 ^ x00, bin ^ F ^ F ^ y01 ^ y00) =
        eq(x01, y01) .and. eq(x00, y00)
    eq(bin, bin ^ y00) = F
    eq(bin, bin ^ y01 ^ y00) = F
    eq(bin, bin ^ y10 ^ y01 ^ y00) = F
    eq(bin, bin ^ F ^ F ^ y01 ^ y00) = F
    eq(bin ^ x00, bin) = F
    eq(bin ^ x00, bin ^ y01 ^ y00) = F
    eq(bin ^ x00, bin ^ y10 ^ y01 ^ y00) = F
    eq(bin ^ x00, bin ^ F ^ F ^ y01 ^ y00) = F
    eq(bin ^ x01 ^ x00, bin) = F
    eq(bin ^ x01 ^ x00, bin ^ y00) = F
    eq(bin ^ x01 ^ x00, bin ^ y10 ^ y01 ^ y00) = F
    eq(bin ^ x01 ^ x00, bin ^ F ^ F ^ y01 ^ y00) = F
    eq(bin ^ x10 ^ x01 ^ x00, bin) = F
    eq(bin ^ x10 ^ x01 ^ x00, bin ^ y00) = F
    eq(bin ^ x10 ^ x01 ^ x00, bin ^ y01 ^ y00) = F
    eq(bin ^ x10 ^ x01 ^ x00, bin ^ F ^ F ^ y01 ^ y00) = F
    eq(bin ^ F ^ F ^ x01 ^ x00, bin) = F
    eq(bin ^ F ^ F ^ x01 ^ x00, bin ^ y00) = F
    eq(bin ^ F ^ F ^ x01 ^ x00, bin ^ y01 ^ y00) = F
    eq(bin ^ F ^ F ^ x01 ^ x00, bin ^ y10 ^ y01 ^ y00) = F
end S11
```

Remark. This specification has also been phrased in the language PERSPECT [9]. Using the tools for PERSPECT it was shown that the specification yields a complete TRS with persistent imports. Moreover due to the fact that PERSPECT does not force the user to provide a name for a variable that occurs in an equation only once, shorter specifications are allowed. Indeed in PERSPECT one can remove the factor *log log n* in the order of the length of specifications of *eqD(n)*.

5.7. A Concise Uniform Specification of eqD(K, n)

Using the specification *S11(n)* of the family *eqD(n)* from Sec. 5.6 above we find a very concise specification *S12(K, n)* for the *eqD(K, n)* without auxiliary sorts as follows: add to *S11(n)* a declaration of the constants in *K* as well as a rewrite rule that rewrites every constant name into a normal form of *S11(n)*, of course different normal forms for different names. The additional length added by these axioms is $(L(K) + \log n + c1)\, n + c2$. Now asymptotically the contribution of *S11(n)* disappears and can be taken into account for by the constant *c1*. Because $\log(n) \leq L(K)$ the bound can be transformed to $(2.L(K) + c1).n + c2'$. It follows that we have found a much more concise specification

than *S7(K, n)*. The only disadvantage of *S12* is that it has no standard normal forms, though its execution is much faster. Indeed the number of reduction steps needed to find a normal form of an expression *eq(a, b)* with *a* and *b* names in *K* is of the order of *log n*.

We expect that this specification of *eqD(K, n)* is asymptotically optimal. Getting the constant down below *2* seems to be a very unnatural matter because one needs all names to occur at least once. Moreover in a concise specification these names must be transformed in to a mathematically useful co-ordinate system. But the elements of that space (covering *D(n)*) must have lengths up to *log n* at least.

6. Conclusions

(i) Every uniform algebraic specification of the structures *eqD(K, n)* that we have found has its drawbacks.

(ii) It may even be considered reasonable to include in a specification language special syntax for uniform specification of the family *eqD(K, n)*. Of course it is not reasonable to include special features for the specification of every arbitrary family of algebras, but the *eqD(K, n)* specifications surface in almost every specification of a practical system.

(iii) The most natural specifications of *eqD(K, n)* involve the natural numbers as an auxiliary sort as well as some additional functions. A uniform specification of size $(2.L(K) + c1).n + c2$ that constitutes a complete TRS can be found. The normalisation of expressions can be very inefficient however (see *S5* in sec 3.2).

(iv) The best specification providing a complete TRS as well as fast termination takes size $(L(K) + log(n) + c1).n + c2$. This specification uses a binary representation of the natural numbers as an auxiliary structure (see *S6* in Sec. 3.3).

This specification seems to be workable for practical purposes, it involves no significant overheads that are implied by the focus on algebraic specifications and complete term rewriting systems.

(v) If auxiliary sorts are disallowed a uniform specification with size $(6.L(K) + c1).n + c2$ can be found. This specification provides a complete TRS with slow normalization and standard normal forms. The size of this uniform specification is asymptotically adequate but its practical value seems nonexistent.

(vi) Allowing specifications to be parameterized with functions increases the power of the algebraic specification language.

7. Remaining Problems

(i) Is it possible to find a uniform specification that provides complete term rewriting systems and has a uniform bound on the number of equations? We conjecture that the

answer will be negative, thus finding a significant difference with the case of uniform initial algebra specifications (see Sec. 5.5).

(ii) Can the constant *6* that occurs in the size of the uniform specification *S7* be improved? (Here one insists that the uniform specification provides complete term rewriting systems having standard normal forms and that it uses no auxiliary sorts).

(iii) Can the bound $(\log n)^3 (\log \log n)$ that was found in Sec. 5.6 be improved?

(iv) We have not been able to find any uniform specification, however inefficient, of *eqD(n)* that provides complete term rewriting systems and has a bounded number of equations and avoids the use of auxiliary sorts.

Acknowledgements

We thank Frank Ponsaert of Philips Research in Brussels for showing us a practical specification that suggests an acute need for concise specifications of algebras of the form *eqD(K, n)*. Moreover we acknowledge the technical suggestions of Hans Mulder of the Programming Research Group of the University of Amsterdam.

References

1. J.C.M. Baeten, J.A. Bergstra, J.W. Klop and W.P. Weijland, "Term rewriting systems with rule priorities", *Theor. Comput. Sci.* **67** (1989) 283-301.
2. J.A. Bergstra, J. Heering and P. Klint, *Algebraic Specification*, ACM Press Frontier Series, Addison Wesley, 1989.
3. J.A. Bergstra and J.V. Tucker, "A characterisation of computable data types by means of a finite equational specification method", *Proc. ICALP '80*, Eds. J.W. de Bakker and J. van Leeuwen, Springer LNCS 85, 1980, pp. 76-90.
4. J.A. Bergstra and J.V. Tucker, "The completeness of the algebraic specification methods for computable datatypes", *Inf. Control* **54** (1982) 186-200.
5. H. Ehrig, W. Fey and H. Hansen, "ACT ONE, an algebraic specification language with two levels of semantics", TU Berlin, FB 20, Techn. Report 83-03, 1983.
6. H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specifications, Vol. 1, Equations and Intitial Semantics*, Springer-Verlag, 1985.
7. K. Futatsugi, J.A. Goguen, J.P. Jouannaud and J. Meseguer, "Principles of OBJ2", *Proc. of the 12th POPL*, ACM, 1985, pp. 52-66.
8. S. Mauw, "An algebraic specification of process algebra, including two examples", Report FVI 87-06, University of Amsterdam, 1987.
9. F. Wiedijk, "Voorlopig rapport over de specificatie-taal Perspect", Report P8811, University of Amsterdam 1988. *(in Dutch)*