# Algebraic Data Types and Induction in $\mu$CRL

Jan Friso Groote *
Utrecht University, Department of Philosophy
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
email: jfg@phil.ruu.nl

Jos van Wamel *
University of Amsterdam, Dep. of Mathematics and Computer Science
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
email: vanw@fwi.uva.nl

**Abstract**

In this paper a thorough treatment of elementary data types is given in the specification language $\mu$CRL, which integrates data specification with process specification. Data specification in $\mu$CRL is directly based on the theory of algebraic data types, which is an equational theory, extended with a rule for constructor induction.

Specifications of the following data types and structures are given: booleans, natural numbers, modulo arithmetic, sets, bags, queues, tables, and 'windows'. Moreover many standard functions are specified and useful properties are derived. For most of the data types also alternative induction schemata are derived, and equality functions are provided that identify exactly those terms that are derivably equal.

*1991 Mathematics Subject Classification:* 68Q40, 68Q45, 68Q65.
*1987 CR Categories:* D.3.1, D.3.3, F.4.1.
*Key Words & Phrases:* Abstract Data Types, Algebraic Specification, Induction Principles, Proof Theory, $\mu$CRL.

## 1 Introduction

This paper came forth from the desire to formally verify communication protocols that use non-trivial data types in the process specification language $\mu$CRL, see [GP90, GP91]. As the complexity of the protocols considered increased, it turned out that a lack of standard data types, and more in particular experience with proving properties of such data types, formed a serious handicap. With this paper we attempt to fill this gap.

Process specification in $\mu$CRL is based on the process algebra ACP (Algebra of Communicating Processes, see e.g. [BK84, BW90]), which is particularly suited for formal description of concurrent, communicating systems. Data types in $\mu$CRL are algebraically specified in the standard way, using sorts, functions and axioms (see e.g. [EM85, MG85, Wir90]). The axiomatic style of specifying abstract data types makes it possible to derive properties of data with the inference rules of equational logic, and moreover term rewriting techniques can be applied.

As a result of this purely algebraic character, specifications in $\mu$CRL lend themselves very well for automated proof checking. Some computer checked verifications of $\mu$CRL specifications were made

*Partly supported by the Foundation for Computer Science in the Netherlands (SION) and by the Netherlands Organisation for Scientific Research (NWO).

using the proof system Coq ([Dow93]), which is based on typed $\lambda$-calculus. As examples of such verifications we mention [BG93, GvdP93, KS93]. For a brief overview of $\mu$CRL and its proof theory, and a survey of verifications of communication protocols in it we refer to the paper [GP94]. This paper also contains a discussion of recent developments in proof techniques based on $\mu$CRL.

A number of data specifications in $\mu$CRL is already available from protocol verifications that were made in the past. However, since the main point of such verifications lies at the process behaviour of the systems, no systematic study of standard data types was undertaken. As a result the derived algebraic laws mainly have an ad hoc character. A major disadvantage of this approach is that the efforts made for one verification can hardly be carried over to another, although often the same data specifications are at stake.

What we do in this paper is collect a number of specifications of standard data types that proved to be useful in the past, or that very probably will be useful for future work. An important question is which equations are interesting enough to include in an enumeration. This is mainly a pragmatic matter. We try to include for each data type the most characteristic algebraic laws, such as commutativity and associativity of functions. The same holds for distributivity properties, absorption, idempotency, etc. The generality we aim at when deriving laws from a specification contains that any equation or *property formula* that can be derived for all terms is given priority over an instance of such an equation or property formula. For example, it is easy to derive for the natural numbers that $n + 0 = 0 + n$, but going one step further leads us to $n + m = m + n$. Since the first equation is an instance of the second, we prefer to include the second one and not the first.

In the proof theory, defined in [GP91], a rule for *constructor induction* is formulated. This induction rule is based on the principle of structural induction on terms, and it adds great formal proof capacity to the equational theory. It turns out, however, that it is often convenient to have other, for some proofs very useful, induction schemata than the ones that directly follow from the induction rule. Therefore a number of derived, alternative induction schemata is provided for most sorts, and their use is shown in detail. For the derivation of such schemata, we have to some extent to circumvent the formal framework of the proof theory of $\mu$CRL by using second order arguments.

In order to formalise the status of structural induction in $\mu$CRL, we extend the language definition of $\mu$CRL with the key word **cons**. The **cons** section of a specification contains those function symbols that are used for giving inductive proofs. Of course, a property formula that is proved by induction is not automatically valid for all closed terms. Therefore we restict the class of models, such that the soundness of the induction rule of $\mu$CRL is always guaranteed. The formal status of induction in $\mu$CRL can now be fully captured: We modify the induction rule of $\mu$CRL by parameterising it for some sort $S$ with the set of functions of sort $S$, declared with **cons**, instead of a constructor set for $S$.

Because many functions can be defined in an elegant way with conditional constructs, we need predicates on terms for arbitrary sorts $S$. It is evident that the 'basic' predicate for all sorts is an equality function $eq : S \times S \to \mathbf{Bool}$, defined in such a way that *derived* or *semantic* equality and equality as specified by $eq$ is equivalent, i.e.

$$(*) \quad s = t \leftrightarrow eq(s,t) = \mathsf{t},$$

where we require that $eq(s,t)$ can be equationally computed for all closed $s, t : S$. We give such functions and derive $(*)$ for all data types that we study. It is found that such derivations are considerably easier to make with alternative induction schemata that are based on induction on two variables.

**Related work.** The idea of distinguishing between constructors and functions is quite common in type theory and typed functional programming languages. In algebraic specification theory, the notion of a **cons** section is also used, see e.g. [Wir90]. A proposal for the introduction of a **cons** section in data specifications in $\mu$CRL is due to [Kam93].

Equality functions that satisfy $(*)$ have received considerable attention in the literature on algebraic specification. It is a fact that there is a finite axiomatisation for $eq$, such that $(*)$ holds, if and only if the data type associated with $S$ is *computable* (see e.g. [BT82]), provided that the specification considered is *consistent*, i.e. $\mathsf{t} = \mathsf{f}$ is not derivable. Moreover we would like to mention that from the perspective of term rewriting, algebraic specifications with *eq* functions that satisfy $(*)$ have special interest; using such *eq* functions it is possible to (automatically) prove identities that cannot be proved by purely equational reasoning, without using structural induction. See e.g. [Gog80], compare this method with [HH82], where a method based on term rewriting techniques is provided for giving inductive proofs. In [BG93] it is shown that without loss of generality we may assume that for any sort $S$ there is a function $\overline{eq} : S \times S \to \mathbf{Bool}$ that satisfies $s = t \leftrightarrow \overline{eq}(s,t) = \mathsf{t}$. This fact turns out to be useful whenever structures over arbitrary sorts are considered, such as queues or sets. Existence of such functions does not imply, however, that (in)equality can be equationally decided.

There is a vast amount of literature on algebraic specification, and much inspiration can be drawn from existing specifications. For useful collections of standard specifications we refer to [Wie90] and the standard texts such as [EM85, MG85, Wir90]. The last reference also contains a large bibliography with publications on algebraic specification. A text that also contains material on proving properties of data from a formal specification is [TvD88]. Although we found quite some material on inductive proofs based on term rewriting, we were surprised to find that our work seems to lay somewhere between the *theory* of algebraic specification and the *application* of it, e.g. in standard libraries, and that we were not able to not find much work in the same field.

**Acknowledgements.** We would like to thank Jan Bergstra, Inge Bethke, Willem Jan Fokkink, and Alban Ponse for having contributed to this paper in various ways.

# Contents

# 2 Data specification in $\mu$CRL

In this section we give some basic material on algebraic specification theory, and more in particular how this theory is incorporated in $\mu$CRL. We start with a brief introduction to the theory of algebraic data types, and then we give in more detail the basic principles of data specification and logical deduction in $\mu$CRL. For a thorough discussion of algebraic specification theory the reader is referred to basic texts, such as [EM85, MG85, Wir90]. Details concerning the syntax of specifications in $\mu$CRL can be found in [GP90], and for the proof theory of $\mu$CRL we refer to [GP91, GP94]. For reasons of readability we do not always strictly follow the syntax of $\mu$CRL in the following sections.

## 2.1 Algebraic specification

An *algebraic specification* $(\Sigma, E)$ consists of:

- A many-sorted *signature* $\Sigma = (\mathcal{S}, F)$, where $\mathcal{S}$ is a set of type or sort names, and $F$ a set of function symbols. A function $f$ of some sort $S$, and with arity $n$, is typed by $f : S_1 \times \ldots \times S_n \to S$, where $S_1, \ldots, S_n$ are the sorts of the arguments of $f$, and $S$ the sort of $f$.

- A set $E$ of $\Sigma$-*equations*, which are expressions of the form $s = t$, where $s$ and $t$ are equally typed terms constructed from variables and function symbols in the usual way. Variables are typed, and they are implicitly universally quantified in all expressions.

The specification $(\Sigma, E)$ defines an *equational theory* consisting of all equations derivable from $E$ by means of *many-sorted equational logic*. We assume that $\Sigma$ is inhabited, i.e. that there are closed terms of every sort $S \in \mathcal{S}$, otherwise unsound deductions would be possible (a closed term is a term without variables). The axioms and rules of inference of the equational theory determined by $(\Sigma, E)$ are listed in Table 1. Substitutions are defined as usual.

$$s = t \qquad \text{for every equation } s = t \in E$$

$$\frac{s = t}{\sigma(s) = \sigma(t)} \qquad \text{for every substitution } \sigma$$

$$\frac{s_1 = t_1 \ldots s_n = t_n}{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)} \qquad \text{for every } n\text{-ary function } f \in F$$

$$s = s \qquad \frac{s = t}{t = s} \qquad \frac{s = t \quad t = u}{s = u}$$

Table 1: Rules for deductions in equational logic

It is convention to write $(\Sigma, E) \vdash s = t$ to express that $s = t$ is derivable in the equational theory determined by $(\Sigma, E)$. In the sequel we mostly just write $s = t$ and take care that the context is clear.

In data specifications in $\mu$CRL sorts are declared using the key word **sort**, functions with **cons** and **func**, variables with **var**, and equations with **rew**. These data specifications are completely in the style of algebraic specification theory. The axioms and rules of inference, however, are regarded from a more general point of view: the language of equations from data specifications is generalised to the so-called *property formulas* of $\mu$CRL.

## 2.2 Properties and property formulas

Properties of data specifications in $\mu$CRL can be formally expressed by a language of properties. These properties consist of identities between data, linked together with propositional connectives. [1]

**Definition 2.1.** Let $V$ be a set of *variable declarations* of the form $\langle x : S \rangle$. A *property over* $((\Sigma, E), V)$ is defined inductively in the following way:

- $\mathcal{F}$ ("Falsum") is a *property over* $((\Sigma, E), V)$,
- $s = t$ is a *property over* $((\Sigma, E), V)$ iff $s = t \in E$,
- $\neg(\phi)$ is a *property over* $((\Sigma, E), V)$ iff $\phi$ is,
- $(\phi \circ \psi)$ with $\circ \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ is a *property over* $((\Sigma, E), V)$ iff both $\phi$ and $\psi$ are.

$\square$

In properties we omit brackets according to the convention that $=$ binds stronger than any of the logical operators, that $\neg$ binds stronger than any of the binary logical operators, and that $\vee, \wedge$ bind stronger than $\rightarrow, \leftrightarrow$. More logical symbols are used than strictly necessary. The symbols $\rightarrow$ and $\mathcal{F}$ are regarded as basic, the other symbols are abbreviations in the usual way (see e.g. [Dal83]).

Now a property formula, the basic means to express that a specification has some algebraic property, simply consists of a property that has the originating specification and the set of variables as an attribute:

$$\phi \textbf{ from } ((\Sigma, E), V),$$

where $\phi$ is a property over $((\Sigma, E), V)$. A property formula $\phi$ **from** $((\Sigma, E), V)$ is called *closed* iff $\phi$ contains no variables from $V$. In the following sections we omit **from** $((\Sigma, E), V)$ from property formulas, and take care that the context is clear.

Over this property language a proof system is defined. It is given in a natural deduction format. It contains so-called 'logical' axioms and rules, suitable to derive the fundamental properties induced by the equality relation $=$ and the propositional connectives. This set-up is based on [TvD88]. A deduction can be seen as a tree of which each node is labelled with a property formula (and possibly the name of a rule which has been applied to obtain the property formula). The leaves of the tree are the *assumptions* (also called hypotheses) of the deduction. Symbols $\mathcal{D}$ are used for arbitrary deductions. The deduction

$$\begin{array}{c} \mathcal{D} \\ \psi \textbf{ from } ((\Sigma, E), V) \end{array}$$

indicates that $\mathcal{D}$ has *conclusion* $\psi$ **from** $((\Sigma, E), V)$.

We use the notation $[\phi$ **from** $((\Sigma, E), V)]$ for a possibly empty set of occurrences of a property formula $\phi$ **from** $((\Sigma, E), V)$ in a deduction, thus

$$\begin{array}{c} [\phi \textbf{ from } ((\Sigma, E), V)] \\ \mathcal{D} \end{array}$$

is a deduction $\mathcal{D}$ with a set $[\phi$ **from** $((\Sigma, E), V)]$ of assumptions in $\mathcal{D}$. As a rule it is assumed that $[\phi$ **from** $((\Sigma, E), V)]$ refers to *all* assumptions of the form $\phi$ **from** $((\Sigma, E), V)$ in $\mathcal{D}$.

Logical deductions are defined inductively.

[1] This and the following subsection are based on Section 3 from [GP94]. The most important omission we make is that we do not treat property formulas with process expressions.

**Definition 2.2.** The single-node tree with a property formula $\phi$ **from** $((\Sigma, E), V)$ as label is a deduction from the open assumption $\phi$ **from** $((\Sigma, E), V)$. There are no cancelled assumptions.

Let $\mathcal{D}_1, \mathcal{D}_2$ be deductions. A new deduction can be constructed according to the rules in Table 2. These rules are subject to the following restrictions:

1. In applications of the introduction rule $\rightarrow$I and the rule RAA (Reductio Ad Absurdum) *all* open assumptions of the form indicated by $[\ldots]$ are cancelled.
2. In applications of $\rightarrow$I, RAA, the reflexivity rule REFL, the variable rule VAR and the substitution rule SUB, the conclusion must be a property formula.
3. In applications of SUB the variable $x$ may not be free in any (uncancelled) hypothesis of $\mathcal{D}_1$.
4. Each application of VAR is restricted to $V \subseteq V'$ or $V' \subseteq V$.

□

$$\dfrac{\begin{matrix}[\phi \textbf{ from } ((\Sigma, E), V)] \\ \mathcal{D}_1 \\ \psi \textbf{ from } ((\Sigma, E), V)\end{matrix}}{\phi \rightarrow \psi \textbf{ from } ((\Sigma, E), V)} \rightarrow\mathrm{I}$$

$$\dfrac{\begin{matrix}\mathcal{D}_1 \\ \phi \textbf{ from } ((\Sigma, E), V)\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \phi \rightarrow \psi \textbf{ from } ((\Sigma, E), V)\end{matrix}}{\psi \textbf{ from } ((\Sigma, E), V)} \rightarrow\mathrm{E}$$

$$\dfrac{\begin{matrix}[\neg\phi \textbf{ from } ((\Sigma, E), V)] \\ \mathcal{D}_1 \\ \mathcal{F} \textbf{ from } ((\Sigma, E), V)\end{matrix}}{\phi \textbf{ from } ((\Sigma, E), V)} \mathrm{RAA} \qquad \dfrac{}{t = t \textbf{ from } ((\Sigma, E), V)} \mathrm{REFL}$$

$$\dfrac{\begin{matrix}\mathcal{D}_1 \\ \phi[s/x] \textbf{ from } ((\Sigma, E), V)\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ s = t \textbf{ from } ((\Sigma, E), V)\end{matrix}}{\phi[t/x] \textbf{ from } ((\Sigma, E), V)} \mathrm{REPL}$$

$$\dfrac{\begin{matrix}\mathcal{D}_1 \\ \phi \textbf{ from } ((\Sigma, E), V)\end{matrix}}{\phi[t/x] \textbf{ from } ((\Sigma, E), V)} \mathrm{SUB} \qquad \dfrac{\begin{matrix}\mathcal{D}_1 \\ \phi \textbf{ from } ((\Sigma, E), V)\end{matrix}}{\phi \textbf{ from } ((\Sigma, E), V')} \mathrm{VAR}$$

Table 2: Rules for logical deductions in $\mu$CRL

The notation $\phi[t/x]$ means that $t$ is substituted for $x$ in $\phi$. The reflexivity rule REFL has an empty premiss, and is therefore called an axiom. The rule VAR is a structural rule that allows restricted replacement of variable sets. With VAR variable sets can be obtained that are suitable for further derivations.

The following results are easily derived:

**Proposition 2.3.** *Let $((\Sigma, E), V)$ be given. It holds that*

1. $s = t \rightarrow t = s$,
2. $s = t \wedge t = u \rightarrow s = u$,
3. $s = t \rightarrow u[s/v] = u[t/v]$.

By definition $s = t$ **from** $((\Sigma, E), V)$ for $s = t \in E$, so if we combine the above results with the rule REFL from Table 2 we easily find the axioms and rules of equational logic in Table 1.

## 2.3 Constructor sets and the induction rule of $\mu$CRL

In $\mu$CRL it is required that any model for the data part of a data specification $(\Sigma, E)$ is minimal, i.e., satisfies what is often referred to in the literature as the *no-junk* property. This means that each of its elements corresponds to at least one closed term over the signature $\Sigma$. With this requirement properties of data may be proved by structural induction on one or more variables. In order to formally capture this principle, an induction rule was introduced in the proof theory of $\mu$CRL. It uses the notion of *constructor sets*: We say that $C$ is a constructor set for the sort $S$ iff all functions in $C$ are of the sort $S$, and any closed data term of sort $S$ can be proved equal to a data term that is obtained from applications of the functions in $C$ and possibly terms not of sort $S$ only. In general it is not possible to prove *within* the framework of the proof theory that a given set is a constructor set.

Therefore, in order to further formalise the status of structural induction in $\mu$CRL, we provide a slightly simplified and modified version of the induction rule given in [GP91]. *We modify the induction rule of $\mu$CRL by parameterising it with a set $\overline{C}$ instead of a constructor set $C$, where $\overline{C}$ is the set of functions of some sort $S$, declared with the (new) key word* **cons**. *As for the models for the data part of a specification, we require that they satisfy the no-junk property w.r.t. the terms over the signature defined by the set $\overline{C}$ and the corresponding sorts.*

Assume that for given $((\Sigma, E), V)$ we have that $\{\langle x : S\rangle\} \subseteq V$. Let

$$\overline{C} \stackrel{\text{def}}{=} \{f_j : S_1^j \times \ldots \times S_{l_j}^j \rightarrow S \mid 1 \leq j \leq k \ , \ k > 0 \ , \ l_j \geq 0\}$$

be the set functions of sort $S$, declared with the key word **cons**, of cardinality $k$. We introduce the following induction rule IND($\overline{C}$), which is parameterised with the set $\overline{C}$ (in the following sections we just write IND if $\overline{C}$ is clear from the context). The induction takes place on the variable $x$.

$$\frac{\begin{array}{c}\mathcal{D}_j \\ (\bigwedge_{\sigma \in I_j} \sigma(\phi)) \rightarrow \phi[f_j(z_1^j, \ldots, z_{l_j}^j)/x] \ \textbf{from} \ ((\Sigma, E), V \cup \{\langle z_n^j : S_n^j\rangle \mid 1 \leq n \leq l_j\})\end{array}}{\phi \ \textbf{from} \ ((\Sigma, E), V)} \quad 1 \leq j \leq k$$

where for each $1 \leq j \leq k$ the index set $I_j$ is a set of substitutions satisfying:

$$\sigma \in I_j \quad \Longleftrightarrow \quad \begin{array}{l}\sigma \text{ is the identity, except that it maps } x \text{ to} \\ \text{some element of the same sort from } \{z_n^j \mid 1 \leq n \leq l_j\}.\end{array}$$

Note that it is required that in IND($\overline{C}$) all the variables $x, z_1^j, \ldots, z_{l_j}^j$ are pairwise different for all appropriate $j$. Also note that if $f_j$ is a constant then $I_j$ is the empty set and thus, by convention, the antecedent in the induction hypothesis (i.h.) $\bigwedge_{\sigma \in I_j} \sigma(\phi)$ is satisfied.

# 3 Booleans

The simple, two-valued booleans are regarded as a standard data type in $\mu$CRL, represented by the sort **Bool**. The reason for including the booleans in the standard definition of the language was that this way, the booleans could be axiomatised such that the conditional construct of $\mu$CRL always behaves properly. Analogously to e.g. [MG85, Wir90] we also adopt the axiom $\neg(\mathsf{t} = \mathsf{f})$ in order to be able to axiomatise equality for given sorts in a satisfactory way (see Introduction). In this section we define some basic functions on the booleans.

In $\mu$CRL, the symbols $\neg$, $\wedge$ and $\vee$ are reserved for the language of property formulas. For this reason we use other function names for corresponding operations on the booleans. Booleans are specified straightforwardly by the below specification, and the standard axioms Bool1 and Bool2 of $\mu$CRL are given in Table 3.

| | | |
|---|---|---|
| **sort** | **Bool** | |
| **cons** | $\mathsf{t}, \mathsf{f} : \rightarrow \mathbf{Bool}$ | |
| **func** | $not : \mathbf{Bool} \rightarrow \mathbf{Bool}$ | |
| | $and, or, eq : \mathbf{Bool} \times \mathbf{Bool} \rightarrow \mathbf{Bool}$ | |
| **var** | $b : \mathbf{Bool}$ | |
| **rew** | $not(\mathsf{t}) = \mathsf{f}$ | Bool3 |
| | $not(\mathsf{f}) = \mathsf{t}$ | Bool4 |
| | $and(\mathsf{t}, b) = b$ | Bool5 |
| | $and(\mathsf{f}, b) = \mathsf{f}$ | Bool6 |
| | $or(\mathsf{t}, b) = \mathsf{t}$ | Bool7 |
| | $or(\mathsf{f}, b) = b$ | Bool8 |
| | $eq(\mathsf{t}, \mathsf{t}) = \mathsf{t}$ | Bool9 |
| | $eq(\mathsf{f}, \mathsf{f}) = \mathsf{t}$ | Bool10 |
| | $eq(\mathsf{t}, \mathsf{f}) = \mathsf{f}$ | Bool11 |
| | $eq(\mathsf{f}, \mathsf{t}) = \mathsf{f}$ | Bool12 |

The standard axioms are requirements that a model of a specification in $\mu$CRL must satisfy. According to axiom Bool1, such a model is *boolean preserving*, i.e. the booleans $\mathsf{t}$ and $\mathsf{f}$ are considered different. According to axiom Bool2 there are at most two booleans. This axiom formally expresses the no-junk property of any model with respect to the booleans. (Axiom Bool2 is not necessary for proving properties of data; we only include it for completeness.) It is not hard to see that both the axioms Bool1 and Bool2 are sound in the valid models for the above specification.

| | |
|---|---|
| (Bool1) | $\neg(\mathsf{t} = \mathsf{f})$ |
| (Bool2) | $\neg(b = \mathsf{t}) \rightarrow b = \mathsf{f}$ |

Table 3: The standard axioms of $\mu$CRL for booleans, where $b : \mathbf{Bool}$

For the booleans, a basic form of induction can be derived: a property formula $\phi(b)$, where $b : \mathbf{Bool}$, is proved if $\phi(\mathsf{t})$ and $\phi(\mathsf{f})$.

**Definition 3.1 (Case Distinction).** The following inference rule is referred to as *case distinction on* $b$ : **Bool**:

$$\frac{\phi(\mathsf{t}) \quad \phi(\mathsf{f})}{\phi(b)}$$

**Proposition 3.2.** *Proofs using case distinction are correct.*

**Proof.** Follows easily from the above specification, and the induction rule IND. □

Below a number of well-known property formulas with booleans is listed.

**Proposition 3.3.** *For all* $b, c, d : \mathbf{Bool}$ *we have:*

| | | | |
|---|---|---|---|
| *1.a.* | $and(b,c) = and(c,b)$, | *1.b.* | $or(b,c) = or(c,b)$, |
| *2.a.* | $and(b,b) = b$, | *2.b.* | $or(b,b) = b$, |
| *3.a.* | $and(b, and(c,d)) = and(and(b,c),d)$, | *3.b.* | $or(b, or(c,d)) = or(or(b,c),d)$, |
| *4.a.* | $and(b, or(b,c)) = b$, | *4.b.* | $or(b, and(b,c)) = b$, |
| *5.a.* | $and(or(b,c),d) = or(and(b,d), and(c,d))$, | *5.b.* | $or(and(b,c),d) = and(or(b,d), or(c,d))$, |
| *6.a.* | $not(and(b,c)) = or(not(b), not(c))$, | *6.b.* | $not(or(b,c)) = and(not(b), not(c))$, |
| *7.a.* | $and(b, not(b)) = \mathsf{f}$, | *7.b.* | $or(b, not(b)) = \mathsf{t}$, |
| *8.* | $not(not(b)) = b$. | | |

**Proof.** Follows by case distinction from Bool3 – 8. □

The following proposition relates boolean property formulas with logical connectives to boolean equations without. The first equation states that if two boolean expressions are derivably equal, these two expressions are equal according to the equality function $eq$.

**Proposition 3.4.** *For all* $b, c : \mathbf{Bool}$ *we have:*

1. $b = c \leftrightarrow eq(b,c) = \mathsf{t}$,
2. $\neg(b = c) \leftrightarrow b = not(c)$,
3. $b = \mathsf{t} \wedge c = \mathsf{t} \leftrightarrow and(b,c) = \mathsf{t}$,
4. $b = \mathsf{t} \vee c = \mathsf{t} \leftrightarrow or(b,c) = \mathsf{t}$.

**Proof.** Follows by case distinction from Bool1 – 12. □

For every sort $S$ we assume a *selector* function $if$, which represents an *if – then – else* construct.

| | | |
|---|---|---|
| **func** | $if : \mathbf{Bool} \times S \times S \to S$ | |
| **var** | $s, t : S$ | |
| **rew** | $if(\mathsf{t}, s, t) = s$ | IfS1 |
| | $if(\mathsf{f}, s, t) = t$ | IfS2 |

**Proposition 3.5.** *For all* $b : \mathbf{Bool}$, $t : S$ *we have:*

$$if(b, t, t) = t.$$

**Proof.** By case distinction. □

The above property formula is a nice illustration of the possibilities of equational logic extended with constructor induction; without the induction schema in Definition 3.1 it cannot be derived.

# 4 Natural numbers

The natural numbers are specified below. For reasons of readability we use infix notations for most of the functions. For similar reasons we write $Sn$ for $S(n)$, and 1 for $S(0)$, 2 for $S(S(0))$ etc.

| | | |
|---|---|---|
| **sort** | $Nat$ | |
| **cons** | $0 :\to Nat$ | |
| | $S : Nat \to Nat$ | |
| **func** | $P : Nat \to Nat$ | |
| | $+, \dot{-}, * : Nat \times Nat \to Nat$ | |
| | $eq, \geq, \leq, <, >: Nat \times Nat \to \mathbf{Bool}$ | |
| **var** | $n, m : Nat$ | |
| **rew** | $P(0) = 0$ | Nat1 |
| | $P(Sn) = n$ | Nat2 |
| | $n + 0 = n$ | Nat3 |
| | $n + Sm = S(n + m)$ | Nat4 |
| | $n \dot{-} 0 = n$ | Nat5 |
| | $n \dot{-} Sm = P(n \dot{-} m)$ | Nat6 |
| | $n * 0 = 0$ | Nat7 |
| | $n * Sm = n + (n * m)$ | Nat8 |
| | $eq(0, 0) = \mathsf{t}$ | Nat9 |
| | $eq(0, Sn) = \mathsf{f}$ | Nat10 |
| | $eq(Sn, 0) = \mathsf{f}$ | Nat11 |
| | $eq(Sn, Sm) = eq(n, m)$ | Nat12 |
| | $n \geq 0 = \mathsf{t}$ | Nat13 |
| | $0 \geq Sn = \mathsf{f}$ | Nat14 |
| | $Sn \geq Sm = n \geq m$ | Nat15 |
| | $n \leq m = m \geq n$ | Nat16 |
| | $n > m = n \geq Sm$ | Nat17 |
| | $n < m = Sn \leq m$ | Nat18 |

We formally formulate the well-known induction schema that can be used for proving properties of natural numbers.

**Definition 4.1 (Standard Induction).** The following inference rule is referred to as *standard induction on* $n : Nat$:

$$\frac{\phi(0) \quad \phi(m) \to \phi(Sm)}{\phi(n)}$$

where $m : Nat$.

**Proposition 4.2.** *Proofs using standard induction are correct.*

**Proof** It can easily be shown that the above schema follows from the induction rule IND, parameterised with the set $\overline{C} = \{0 :\to Nat, S : Nat \to Nat\}$. Its soundness then follows from the soundness of IND. In fact, it can also be proved from the above specification that every closed term of sort $Nat$ can be represented as either 0 or $Sm$ for some $m : Nat$, i.e., that the set $\overline{C}$ is indeed a constructor set (a similar result can be obtained for any other data specification in this paper). □

In [TvD88] a system *primitive recursive arithmetic* (PRA) is defined, which is equivalent to the system specified above. The only difference is that we use two notions of equality, expressed by $=$ and $eq$, which are equivalent according to Proposition 4.5. In the aforementioned work also a schema for *double induction* is derived. It turns out that many properties of natural numbers are considerably easier to prove when double induction is used instead of standard induction.

**Definition 4.3 (Double Induction).** The following inference rule is referred to as *double induction on* $m, n : Nat$:

$$\frac{\phi(k,0) \quad \phi(0,l) \quad \phi(k,l) \rightarrow \phi(Sk, Sl)}{\phi(m,n)}$$

where $k, l : Nat$.

**Proposition 4.4.** *Proofs using double induction are correct.*

**Proof.** Follows from the correctness of standard induction, see e.g. [TvD88]. □

As for special induction schemata we have the following two remarks: 1) The double induction schema is not derivable *within* the proof theory of $\mu$CRL, just as some other schemata that we give in the sequel, since such a derivation is 'infinite': for every pair of terms $m_0, n_0 : Nat$ there is a different proof of $\phi(m_0, n_0)$. 2) For the derivation of such alternative schemata we will have to use some second order arguments, i.e. we have to reason about property formulas, using variables $\phi, \psi, \ldots$ ranging over some universe of property formulas.

In order to demonstrate the use of both induction schemata we prove the following important result in detail. It states that derivable equality of natural numbers is equivalent to equality according to the *eq* function.

**Proposition 4.5 (Equality).** *For all* $m, n : Nat$ *we have:*

$$m = n \;\leftrightarrow\; eq(m,n) = \mathsf{t}.$$

**Proof.** See Appendix. □

Below a number of property formulas with natural numbers is listed.

**Proposition 4.6.** *For all* $k, l, m, n : Nat$ *we have:*

1. $m + n = n + m$,
2. $k + (m + n) = (k + m) + n$,
3. $0 \dot{-} n = 0$,
4. $(m + n) \dot{-} n = m$,
5. $k \dot{-} (m + n) = (k \dot{-} m) \dot{-} n$,
6. $m * n = n * m$,
7. $k * (m * n) = (k * m) * n$,
8. $k * (m + n) = (k * m) + (k * n)$,
9. $k * (m \dot{-} n) = (k * m) \dot{-} (k * n)$,
10. $eq(m + n, 0) = and(eq(m,0), eq(n,0))$,
11. $eq(m + k, n + k) = eq(m,n)$,
12. $eq(m \dot{-} n, 0) = n \geq m$,

13. $n \geq n = \mathsf{t}$,
14. $m \geq n = or(m > n, eq(m, n))$,
15. $m > n = and(m \geq n, not(eq(m, n)))$,
16. $m > n = m \dot{-} n > 0$,
17. $n > n = \mathsf{f}$,
18. $not(m \geq n) = m < n$,
19. $and(m \geq n, n > m) = \mathsf{f}$,
20. $or(m \geq n, n > m) = \mathsf{t}$,
21. $or(m > n, m < n) = not(eq(m, n))$,
22. $m < n = n > m$,
23. $m > n = \mathsf{t} \rightarrow m \geq n = \mathsf{t}$,
24. $m \geq n = \mathsf{t} \wedge k > 0 = \mathsf{t} \rightarrow k + m > n = \mathsf{t}$,
25. $m > n = \mathsf{t} \rightarrow k + m > n = \mathsf{t}$,
26. $k \geq m = \mathsf{t} \rightarrow k \geq m \dot{-} n = \mathsf{t}$,
27. $k > m = \mathsf{t} \rightarrow k > m \dot{-} n = \mathsf{t}$,
28. $k \geq m = \mathsf{t} \wedge l \geq n = \mathsf{t} \rightarrow k + l \geq m + n = \mathsf{t}$,
29. $k \geq m = \mathsf{t} \wedge l \leq n = \mathsf{t} \rightarrow k \dot{-} l \geq m \dot{-} n = \mathsf{t}$,
30. $k \geq m = \mathsf{t} \wedge m \geq n = \mathsf{t} \rightarrow k \geq n = \mathsf{t}$,
31. $k \geq m = \mathsf{t} \wedge m > n = \mathsf{t} \rightarrow k > n = \mathsf{t}$,
32. $k > m = \mathsf{t} \wedge m \geq n = \mathsf{t} \rightarrow k > n = \mathsf{t}$,
33. $k > m = \mathsf{t} \wedge m > n = \mathsf{t} \rightarrow k > Sn = \mathsf{t}$,
34. $m \geq n = \mathsf{t} \rightarrow eq(k + n, m) = eq(k, m \dot{-} n)$,
35. $m > k = \mathsf{t} \rightarrow (k + n) \dot{-} m < n = 0 < n$,
36. $m > 0 = \mathsf{t} \rightarrow k < m + n = k \dot{-} n < m$,
37. $m \geq n = \mathsf{t} \wedge m \dot{-} n = 0 \rightarrow m = n$,
38. $k \geq m = \mathsf{t} \rightarrow (k \dot{-} m) + n = (k + n) \dot{-} m$,
39. $k \geq m = \mathsf{t} \wedge m \geq n = \mathsf{t} \rightarrow (k \dot{-} m) + n = k \dot{-} (m \dot{-} n)$,
40. $k \geq m \dot{-} n = \mathsf{t} \wedge m \geq n = \mathsf{t} \rightarrow k \dot{-} (m \dot{-} n) = (k + n) \dot{-} m$.

**Proof.** Proofs of the above property formulas are standard using the given induction schemata for natural numbers. However, not all proofs are easy; we refer to [TvD88] for more details. □

## 5 Modulo arithmetic

Modulo addition and subtraction are specified with the help of the function $.\,mod\,m : Nat \to Nat$, which can be regarded as a unary function that induces an equivalence relation on the sort $Nat$ for each $m > 0$.

| | | |
|---|---|---|
| **func** | $mod : Nat \times Nat \to Nat$ | |
| | $+, - : Nat \times Nat \times Nat \to Nat$ | |
| **var** | $k, m, n : Nat$ | |
| **rew** | $m\,mod\,0 = m$ | Mod1 |
| | $m\,mod\,Sn = if(m \geq Sn, (m \dot{-} Sn)\,mod\,Sn, m)$ | Mod2 |
| | $k +_n m = (k + m)\,mod\,n$ | Mod3 |
| | $k -_n m = if(k\,mod\,n \geq m\,mod\,n, k\,mod\,n \dot{-} m\,mod\,n, n \dot{-} (m\,mod\,n \dot{-} k\,mod\,n))$ | Mod4 |

The $.\,mod\,m$ function is specified iteratively using the selector function $if$. Note that the form of the rewrite rules enforces us to prove property formulas with expressions $k\,mod\,m$ for both $m \equiv 0$ and $m \equiv Sn$.

It turns out that straightforward application of the constructor induction schemata from the definitions 4.1 and 4.3 often leads to cumbersome calculations. The following induction schema sometimes leads to much simpler proofs.

**Definition 5.1 (Special Induction 1).** Let $n : Nat$ with $n > 0 = \mathsf{t}$. The following inference rule is referred to as *special induction 1 on* $p : Nat$:

$$\frac{k < n = \mathsf{t} \to \phi(k) \qquad \phi(m) \to \phi(m+n)}{\phi(p)}$$

where $k, m : Nat$.

**Proposition 5.2.** *Proofs using special induction 1 are correct.*

**Proof.** See Appendix. □

**Definition 5.3 (Special Induction 2).** Let $n : Nat$ with $n > 0 = \mathsf{t}$. The following inference rule is referred to as *special induction 2 on* $m : Nat$:

$$\frac{k < n = \mathsf{t} \to \phi(k)}{\phi(m\,mod\,n)}$$

where $k : Nat$.

**Proposition 5.4.** *Proofs using special induction 2 are correct.*

**Proof.** Follows easily from the previous special induction schema and axiom Mod2. □

The following basic property formulas for the $mod$ function can now be proved in a simple way.

**Proposition 5.5.** *For all $k, m, n : Nat$ we have:*

1. $0\,mod\,m = 0$,
2. $m\,mod\,1 = 0$,
3. $(k + (m * n))\,mod\,n = k\,mod\,n$,

4. $(k + (m \, mod \, n)) \, mod \, n = (k + m) \, mod \, n$,
5. $(k * (m \, mod \, n)) \, mod \, n = (k * m) \, mod \, n$,
6. $m \, mod \, n \leq m = \mathsf{t}$,
7. $m \, mod \, Sn < Sn = \mathsf{t}$.

**Proof.** See Appendix. □

We now study the modulo addition $. +_n .$ and subtraction $. -_n .$ more closely. Observe that these functions can be interpreted as functions on equivalence classes induced by $. \, mod \, n$: it follows easily from axiom Mod3 and Proposition 5.5.4 that

$$(k \, mod \, n) +_n m = k +_n (m \, mod \, n) = k +_n m,$$

and from axiom Mod4 it follows that

$$(k \, mod \, n) -_n m = k -_n (m \, mod \, n) = k -_n m.$$

From now on we use $k_n, l_n, \ldots$ as abbreviations for $k \, mod \, n, l \, mod \, n, \ldots$, and $Sk_n, Sl_n \ldots$, to denote $(Sk)_n, (Sl)_n, \ldots$ Some standard results for modulo addition and subtraction can be derived.

**Proposition 5.6.** *For all $k, l, m, n : Nat$ we have:*

1. $k +_0 m = k + m$,
2. $k -_0 m = k \dot{-} m$,
3. $k +_n m = m +_n k$,
4. $k +_n (l +_n m) = (k +_n l) +_n m$,
5. $k -_{Sn} (l +_{Sn} m) = (k -_{Sn} l) -_{Sn} m$,
6. $k +_{Sn} (l -_{Sn} m) = (k +_{Sn} l) -_{Sn} m$,
7. $k -_{Sn} (l -_{Sn} m) = (k -_{Sn} l) +_{Sn} m$.

**Proof.** See Appendix. □

The following proposition contains some property formulas that express how the relational functions $eq$, $<$, $\leq$, etc. interact with modulo arithmetic.

**Proposition 5.7.** *For all $k, l, m, n : Nat$ we have:*

1. $k +_{Sn} m < Sn = \mathsf{t}$,
2. $k -_{Sn} m < Sn = \mathsf{t}$,
3. $eq(k +_n l, m +_n l) = eq(k_n, m_n)$,
4. $eq(k -_{Sn} l, m -_{Sn} l) = eq(k_{Sn}, m_{Sn})$,
5. $eq(k +_{Sn} m, 0) = or(and(eq(k_{Sn}, 0), eq(m_{Sn}, 0)), eq(k_{Sn}, Sn \dot{-} m_{Sn}))$,
6. $eq(k -_{Sn} m, 0) = eq(k_{Sn}, m_{Sn})$,
7. $k +_{Sn} m < m_{Sn} = and(k_{Sn} \geq Sn \dot{-} m_{Sn}, m_{Sn} > 0)$,
8. $k -_{Sn} m < Sn -_{Sn} m = and(k_{Sn} \geq m_{Sn}, m_{Sn} > 0)$,

9. $l_{Sn} < m_{Sn} = \mathsf{t} \rightarrow k -_{Sn} l < m -_{Sn} l = and(l_{Sn} \leq k_{Sn}, k_{Sn} < m_{Sn})$.

**Proof.** See Appendix. □

We give another induction schema that will sometimes make it easier to prove property formulas for modulo arithmetic that have more than one variable of the sort *Nat*.

**Definition 5.8 (Modulo Induction).** Let $q : Nat$, and $n : Nat$ with $n > 0 = \mathsf{t}$. The following inference rules are referred to as *modulo induction on* $p : Nat$:

1. $$\frac{\phi(m_n, m_n) \quad \phi(k_n, m_n) \rightarrow \phi(Sk_n, m_n)}{\phi(p_n, q_n)} \quad or$$

2. $$\frac{\phi(m_n, m_n) \quad \phi(Sk_n, m_n) \rightarrow \phi(k_n, m_n)}{\phi(p_n, q_n)}$$

where $k, m : Nat$.

**Proposition 5.9.** *Proofs using modulo induction are correct.*

**Proof.** See Appendix. □

Note that if we replace the premise $\psi(k_n, m_n)$ in one of the above induction schemata by the stronger $\psi(k_n, m_n) \wedge eq(k_n, m_n) = \mathsf{f}$, we obtain an equivalent schema. This fact will be used in Section 10.

# 6 Sets

Sets with elements of some arbitrary sort $D$ are specified in a standard fashion. We do not need any specific assumptions on $D$, except that there has to be a function $\overline{eq} : D \times D \rightarrow \mathbf{Bool}$ satisfying $d = e \leftrightarrow \overline{eq}(d, e) = \mathsf{t}$ for $d, e : D$, and that $D$ has a special element $d_0$ (although we do not need $d_0$ yet). We said in the Introduction that we may assume a function $\overline{eq}$ without loss of generality. The empty set is represented by the constant $s_0$, and in order to increase readability we use the standard (infix) function symbols $\cup$, $\cap$, and $\setminus$ for representing operations on sets (this is not in accordance with the syntax of $\mu$CRL).

The characteristic properties of sets are captured by axiom Set9, which implies that every element in a set occurs only once, and axiom Set10, according to which there is no specific order in which the elements in a set are arranged. Typical consequences of these axioms are respectively idempotency (Proposition 6.4.1), and commutativity (Proposition 6.4.2) of the operators $\cup$ and $\cap$.

| | |
|---|---|
| **sort** | *Set* |
| **cons** | $s_0 :\rightarrow Set$ |
| | $in : D \times Set \rightarrow Set$ |
| **func** | $rem : D \times Set \rightarrow Set$ |
| | $\cup, \cap, \setminus : Set \times Set \rightarrow Set$ |
| | $eq, \subset, \subseteq : Set \times Set \rightarrow \mathbf{Bool}$ |
| | $empty : Set \rightarrow \mathbf{Bool}$ |
| | $test : D \times Set \rightarrow \mathbf{Bool}$ |
| | $size : Set \rightarrow Nat$ |
| **var** | $d, e : D$ |
| | $s, t : Set$ |

| | | |
|---|---|---|
| **rew** | $rem(d, s_0) = s_0$ | Set1 |
| | $rem(d, in(e, s)) = if(\overline{eq}(d, e), rem(d, s), in(e, rem(d, s)))$ | Set2 |
| | $s_0 \cup s = s$ | Set3 |
| | $in(d, s) \cup t = in(d, s \cup t)$ | Set4 |
| | $s_0 \cap s = s_0$ | Set5 |
| | $in(d, s) \cap t = if(test(d, t), in(d, s \cap t), s \cap t)$ | Set6 |
| | $s \setminus s_0 = s$ | Set7 |
| | $s \setminus in(d, t) = rem(d, s \setminus t)$ | Set8 |
| | $in(d, in(d, s)) = in(d, s)$ | Set9 |
| | $in(d, in(e, s)) = in(e, in(d, s))$ | Set10 |
| | $eq(s_0, s_0) = \mathsf{t}$ | Set11 |
| | $eq(s_0, in(d, s)) = \mathsf{f}$ | Set12 |
| | $eq(in(d, s), t) = and(test(d, t), eq(rem(d, s), rem(d, t)))$ | Set13 |
| | $s_0 \subset s_0 = \mathsf{f}$ | Set14 |
| | $s_0 \subset in(d, s) = \mathsf{t}$ | Set15 |
| | $in(d, s) \subset t = and(test(d, t), rem(d, s) \subset rem(d, t))$ | Set16 |
| | $s \subseteq t = or(eq(s, t), s \subset t)$ | Set17 |
| | $empty(s) = eq(size(s), 0)$ | Set18 |
| | $test(d, s_0) = \mathsf{f}$ | Set19 |
| | $test(d, in(e, s)) = or(\overline{eq}(d, e), test(d, s))$ | Set20 |
| | $size(s_0) = 0$ | Set21 |
| | $size(in(d, s)) = size(rem(d, s)) + 1$ | Set22 |

**Definition 6.1 (Standard Set Induction).** The following inference rule is referred to as *standard set induction on $s : Set$*:

$$\frac{\phi(s_0) \quad \phi(t) \to \phi(in(d,t))}{\phi(s)}$$

where $d : D$ and $t : Set$.

**Proposition 6.2.** *Proofs using standard set induction are correct.*

**Proof.** Follows from the above specification and the induction rule IND. □

We start off with some facts about the most basic operations on sets.

**Proposition 6.3.** *For all $s, t : Set$ and $d, e : D$ we have:*

1. $in(d, rem(d, s)) = in(d, s)$,
2. $rem(d, rem(d, s)) = rem(d, s)$,
3. $rem(d, rem(e, s)) = rem(e, rem(d, s))$,
4. $test(d, s) = \mathsf{t} \to in(d, s) = s$,
5. $test(d, s) = \mathsf{f} \to rem(d, s) = s$,
6. $test(d, rem(d, s)) = \mathsf{f}$,
7. $\overline{eq}(d, e) = \mathsf{f} \to test(d, rem(e, s)) = test(d, s)$,
8. $test(d, s \cup t) = or(test(d, s), test(d, t))$,

9. $test(d, s \cap t) = and(test(d, s), test(d, t))$,
10. $test(d, t \setminus s) = and(test(d, t), not(test(d, s)))$,
11. $empty(s \cup t) = and(empty(s), empty(t))$,
12. $rem(d, s \cup t) = rem(d, s) \cup rem(d, t)$,
13. $rem(d, s \cap t) = rem(d, s) \cap rem(d, t)$,
14. $rem(d, t \setminus s) = rem(d, t) \setminus rem(d, s)$.

**Proof.** By induction on $s$. □

Note the close resemblance between the *rem* function (propositions 6.3.2 and 6.3.3), and the *in* function (axioms Set9 and Set10).

Below we list a number of well-known identities for sets, which can be derived quite easily using the previous proposition.

**Proposition 6.4.** *For all $s, t, u : Set$ we have:*

| | | | |
|---|---|---|---|
| *1.a.* | $s \cap s = s$, | *1.b.* | $s \cup s = s$, |
| *2.a.* | $s \cap t = t \cap s$, | *2.b.* | $s \cup t = t \cup s$, |
| *3.a.* | $s \cap (t \cap u) = (s \cap t) \cap u$, | *3.b.* | $s \cup (t \cup u) = (s \cup t) \cup u$, |
| *4.a.* | $s \cap (s \cup t) = s$, | *4.b.* | $s \cup (s \cap t) = s$, |
| *5.a.* | $s \cap (t \cup u) = (s \cap t) \cup (s \cap u)$, | *5.b.* | $s \cup (t \cap u) = (s \cup t) \cap (s \cup u)$, |
| *6.a.* | $(t \cap u) \setminus s = (t \setminus s) \cap (u \setminus s)$, | *6.b.* | $(t \cup u) \setminus s = (t \setminus s) \cup (u \setminus s)$, |
| *7.* | $s \setminus s = s_0$, | | |
| *8.* | $(t \setminus u) \setminus s = t \setminus (u \cup s)$, | | |
| *9.* | $t \cup (t \setminus s) = t$, | | |
| *10.* | $t \setminus (u \cap s) = (t \setminus u) \cup (t \setminus s)$, | | |
| *11.* | $eq(s, s \cap t) = eq(s \cup t, t)$. | | |

**Proof.** By induction on $s$. □

For proving property formulas for sets with boolean functions $eq$, $\subset$ and $\subseteq$, some useful new induction schemata can be introduced.

**Definition 6.5 (Special Set Induction 1).** The following inference rule is referred to as *special set induction 1 on $s : Set$*:

$$\frac{\phi(s_0) \quad \phi(rem(d,t)) \rightarrow \phi(in(d,t))}{\phi(s)}$$

where $d : D$ and $t : Set$.

**Proposition 6.6.** *Proofs by special set induction with the above schema are correct.*

**Proof.** See Appendix. □

**Definition 6.7 (Special Set Induction 2).** Let $f : D \times Set \to Set$ be a function. The following inference rule is referred to as *special set induction 2 on* $v : Set$:

$$\frac{\phi(s_0, t) \quad \phi(rem(d, s), f(d, t)) \to \phi(in(d, s), t)}{\phi(v, w)}$$

where $d : D$ and $s, t, w : Set$.

**Proposition 6.8.** *Proofs by special set induction with the above schema are correct.*

**Proof.** See Appendix. □

The following proposition states that derivable equality of sets is equivalent to equality as specified by the *eq* function. In the proof of this proposition we demonstrate the use of the second special induction schema in detail.

**Proposition 6.9 (Set Equality).** *For all* $s, t : Set$ *we have:*

$$s = t \;\leftrightarrow\; eq(s, t) = \mathsf{t}.$$

**Proof.** See Appendix. □

Now it is not hard to prove some property formulas that contain relational operators.

**Proposition 6.10.** *For all* $s, t, u : Set$ *we have:*

1. $s \subset s = \mathsf{f}$,
2. $s \cap t \subseteq t = \mathsf{t}$,
3. $s \subseteq s \cup t = \mathsf{t}$,
4. $s \setminus t \subseteq s = \mathsf{t}$,
5. $s \subset t = \mathsf{t} \wedge t \subset u = \mathsf{t} \to s \subset u = \mathsf{t}$,
6. $s \subseteq t = \mathsf{t} \wedge t \subset u = \mathsf{t} \to s \subset u = \mathsf{t}$,
7. $s \subset t = \mathsf{t} \wedge t \subseteq u = \mathsf{t} \to s \subset u = \mathsf{t}$,
8. $s \subseteq t = \mathsf{t} \wedge t \subseteq u = \mathsf{t} \to s \subseteq u = \mathsf{t}$.

**Proof.** See Appendix. □

**Definition 6.11 (Special Set Induction 3).** The following inference rule is referred to as *special set induction 3 on* $s : Set$:

$$\frac{\phi(s_0) \quad test(d, t) = \mathsf{t} \wedge \phi(rem(d, t)) \to \phi(t)}{\phi(s)}$$

where $d : D$ and $t : Set$.

**Proposition 6.12.** *Proofs by special set induction with the above schema are correct.*

**Proof.** See Appendix. □

Sometimes it is possible to specify a function $f : Set \to D$ that selects a particular element from a set. E.g. when $D$ is linearly ordered, a function can be specified that picks the maximum or minimum element from a set (see also Section 11 for a special use of ordered sets). In order to anticipate inductive definitions in data or process specifications in $\mu$CRL that use such functions, e.g. of the form

$$F(s) \stackrel{\text{def}}{=} G(F(rem(f(s), s)), f(s)),$$

we incorporate this idea in the following simple induction schema.

**Definition 6.13 (Special Set Induction 4).** Let $f : Set \to D$ be a function with the property $eq(s, s_0) = \mathsf{f} \to test(f(s), s) = \mathsf{t}$. The following inference rule is referred to as *special set induction 4 on $s : Set$*:

$$\frac{\phi(s_0) \quad \phi(rem(f(t), t)) \to \phi(t)}{\phi(s)}$$

where $t : Set$.

**Proposition 6.14.** *Proofs by special set induction with the above schema are correct.*

**Proof.** Follows from the previous induction schema. □

# 7 Bags

Bags are structures that can be specified quite similar to sets. The composition of bags, defined by the *con* operator, is commutative, which is a result of axiom Bag7 (note that axiom Bag7 is an analogon of axiom Set10 for the sets). The most characteristic difference with sets, however, is that there is no limit to the number of occurences of a certain data element in a bag. So there is no analogon of axiom Set9, and this fact is also reflected in the specification of the *rem* function, which removes at most one element from a bag. The function *diff* defines a difference function between two bags. We regard its status as auxiliary, so we will not study its properties very closely. The empty bag is represented by the constant $b_0$.

| | | |
|---|---|---|
| **sort** | $Bag$ | |
| **cons** | $b_0 :\to Bag$ | |
| | $in : D \times Bag \to Bag$ | |
| **func** | $rem : D \times Bag \to Bag$ | |
| | $con, diff : Bag \times Bag \to Bag$ | |
| | $eq : Bag \times Bag \to \mathbf{Bool}$ | |
| | $empty : Bag \to \mathbf{Bool}$ | |
| | $test : D \times Bag \to \mathbf{Bool}$ | |
| | $size : Bag \to Nat$ | |
| **var** | $d, e : D$ | |
| | $b, c : Bag$ | |
| **rew** | $rem(d, b_0) = b_0$ | Bag1 |
| | $rem(d, in(e, b)) = if(\overline{eq}(d, e), b, in(e, rem(d, b)))$ | Bag2 |
| | $con(b_0, b) = b$ | Bag3 |
| | $con(in(d, b), c) = in(d, con(b, c))$ | Bag4 |

$$
\begin{array}{lr}
\mathit{diff}(b, b_0) = b & \text{Bag5} \\
\mathit{diff}(b, in(d, c)) = rem(d, \mathit{diff}(b, c)) & \text{Bag6} \\
in(d, in(e, b)) = in(e, in(d, b)) & \text{Bag7} \\
eq(b_0, b_0) = \mathsf{t} & \text{Bag8} \\
eq(b_0, in(d, b)) = \mathsf{f} & \text{Bag9} \\
eq(in(d, b), c) = and(test(d, c), eq(b, rem(d, c))) & \text{Bag10} \\
empty(b) = eq(size(b), 0) & \text{Bag11} \\
test(d, b_0) = \mathsf{f} & \text{Bag12} \\
test(d, in(e, b)) = or(\overline{eq}(d, e), test(d, b)) & \text{Bag13} \\
size(b_0) = 0 & \text{Bag14} \\
size(in(d, b)) = size(b) + 1 & \text{Bag15}
\end{array}
$$

**Definition 7.1 (Standard Bag Induction).** The following inference rule is referred to as *standard bag induction on* $b : Bag$:

$$\frac{\phi(b_0) \quad \phi(c) \to \phi(in(d, c))}{\phi(b)}$$

where $d : D$ and $c : Bag$.

**Proposition 7.2.** *Proofs using standard bag induction are correct.*

**Proof.** Follows from the above specification and the induction rule IND. □

We have the following basic facts about bags.

**Proposition 7.3.** *For all* $a, b, c : Bag$ *and* $d : D$ *we have:*

1. $con(b, c) = con(c, b)$,
2. $con(a, con(b, c)) = con(con(a, b), c)$,
3. $size(con(b, c)) = size(b) + size(c)$,
4. $test(d, con(b, c)) = or(test(d, b), test(d, c))$,
5. $empty(con(b, c)) = and(empty(b), empty(c))$,
6. $test(d, b) = \mathsf{t} \to b = in(d, rem(d, b))$.

**Proof.** See Appendix. □

In order to prove that derivable equality is equivalent to equality as specified by the *eq* function for bags, we can use a slightly more general induction schema than the standard one.

**Definition 7.4 (Special Bag Induction 1).** Let $f : D \times Bag \to Bag$ be a function. The following inference rule is referred to as *special bag induction 1 on* $v : Bag$:

$$\frac{\phi(b_0, c) \quad \phi(b, f(d, c)) \to \phi(in(d, b), c)}{\phi(v, w)}$$

where $d : D$ and $b, c, w : Bag$.

**Proposition 7.5.** *Proofs by special bag induction with the above schema are correct.*

**Proof.** See Appendix. □

**Proposition 7.6 (Bag Equality).** *For all $b, c : Bag$ we have:*

$$b = c \;\leftrightarrow\; eq(b,c) = \mathsf{t}.$$

**Proof.** See Appendix. □

We conclude this section with two alternative induction schemata.

**Definition 7.7 (Special Bag Induction 2).** The following inference rule is referred to as *special bag induction 2 on $b : Bag$*:

$$\frac{\phi(b_0) \quad test(d,c) = \mathsf{t} \wedge \phi(rem(d,c)) \to \phi(c)}{\phi(b)}$$

where $d : D$ and $c : Bag$.

**Proposition 7.8.** *Proofs by special bag induction with the above schema are correct.*

**Proof.** Just as the proof of Proposition 6.12. □

**Definition 7.9 (Special Bag Induction 3).** Let $f : Bag \to D$ be a function with the property $eq(b, b_0) = \mathsf{f} \to test(f(b), b) = \mathsf{t}$. The following inference rule is referred to as *special bag induction 3 on $b : Bag$*:

$$\frac{\phi(b_0) \quad \phi(rem(f(c),c)) \to \phi(c)}{\phi(b)}$$

where $c : Bag$.

**Proposition 7.10.** *Proofs by bag induction with the above schema are correct.*

**Proof.** Follows from the previous induction schema. □

# 8 Queues

Queues can be specified with the same constructors as sets and bags, i.e., an empty element and a function *in*. The empty queue is represented by $q_0$.

| | |
|---|---|
| **sort** | $Queue$ |
| **cons** | $q_0 :\to Queue$ |
| | $in : D \times Queue \to Queue$ |
| **func** | $rem : D \times Queue \to Queue$ |
| | $tl, untoe : Queue \to Queue$ |
| | $con : Queue \times Queue \to Queue$ |
| | $hd, toe : Queue \to D$ |
| | $eq : Queue \times Queue \to \mathbf{Bool}$ |
| | $empty : Queue \to \mathbf{Bool}$ |
| | $test : D \times Queue \to \mathbf{Bool}$ |
| | $size : Queue \to Nat$ |
| **var** | $d, e : D$ |
| | $q, r : Queue$ |

**rew**

| | | |
|---|---|---|
| **rew** | $rem(d, q_0) = q_0$ | Queue1 |
| | $rem(d, in(e, q)) = if(\overline{eq}(d, e), q, in(e, rem(d, q)))$ | Queue2 |
| | $tl(q_0) = q_0$ | Queue3 |
| | $tl(in(d, q)) = q$ | Queue4 |
| | $untoe(q_0) = q_0$ | Queue5 |
| | $untoe(in(d, q_0)) = q_0$ | Queue6 |
| | $untoe(in(d, in(e, q))) = in(d, untoe(in(e, q)))$ | Queue7 |
| | $con(q_0, q) = q$ | Queue8 |
| | $con(in(d, q), r) = in(d, con(q, r))$ | Queue9 |
| | $hd(q_0) = d_0$ | Queue10 |
| | $hd(in(d, q)) = d$ | Queue11 |
| | $toe(q_0) = d_0$ | Queue12 |
| | $toe(in(d, q_0)) = d$ | Queue13 |
| | $toe(in(d, in(e, q))) = toe(in(e, q))$ | Queue14 |
| | $eq(q_0, q_0) = \mathsf{t}$ | Queue15 |
| | $eq(q_0, in(d, q)) = \mathsf{f}$ | Queue16 |
| | $eq(in(d, q), q_0) = \mathsf{f}$ | Queue17 |
| | $eq(in(d, q), in(e, r)) = and(\overline{eq}(d, e), eq(q, r))$ | Queue18 |
| | $empty(q) = eq(size(q), 0)$ | Queue19 |
| | $test(d, q_0) = \mathsf{f}$ | Queue20 |
| | $test(d, in(e, q)) = or(\overline{eq}(d, e), test(d, q))$ | Queue21 |
| | $size(q_0) = 0$ | Queue22 |
| | $size(in(d, q)) = size(q) + 1$ | Queue23 |

**Definition 8.1 (Standard Queue Induction).** The following inference rule is referred to as *standard queue induction on $q : Queue$*:

$$\frac{\phi(q_0) \quad \phi(r) \to \phi(in(d, r))}{\phi(q)}$$

where $d : D$ and $r : Queue$.

**Proposition 8.2.** *Proofs using standard queue induction are correct.*

**Proof.** Follows from the above specification and the induction rule IND. □

**Proposition 8.3.** *For all $q, r, s : Queue$ and $d : D$ we have:*

1. $con(q, con(r, s)) = con(con(q, r), s)$,
2. $rem(hd(q), q) = tl(q)$,
3. $size(untoe(q)) = size(q) \dot{-} 1$,
4. $size(tl(q)) = size(q) \dot{-} 1$,
5. $size(con(q, r)) = size(q) + size(r)$,
6. $test(d, con(q, r)) = or(test(d, q), test(d, r))$,
7. $empty(con(q, r)) = and(empty(q), empty(r))$,
8. $eq(q, q_0) = \mathsf{f} \to in(hd(q), tl(q)) = q$.

**Proof.** Easy, by induction on $q$. $\square$

In order to prove that derivable equality is equivalent to equality as specified by the *eq* function for queues, we give two new induction schemata, which are more general than the standard one.

**Definition 8.4 (Special Queue Induction).** Let $f : Queue \to Queue$ be a function. The following inference rule is referred to as *special queue induction on* $v : Queue$:

$$\frac{\phi(q_0, r) \quad \phi(q, f(r)) \to \phi(in(d, q), r)}{\phi(v, w)}$$

where $d : D$ and $q, r, w : Queue$.

**Proposition 8.5.** *Proofs using special queue induction are correct.*

**Proof.** See Appendix. $\square$

Analogously to the natural numbers, we can derive a schema for double induction (see Definition 4.3).

**Definition 8.6 (Double Queue Induction).** The following inference rule is referred to as *double queue induction on* $v, w : Queue$:

$$\frac{\phi(q_0, r) \quad \phi(q, q_0) \quad \phi(q, r) \to \phi(in(d, q), in(e, r))}{\phi(v, w)}$$

where $d, e : D$ and $q, r : Queue$.

**Proposition 8.7.** *Proofs using double queue induction are correct.*

**Proof.** See Appendix. $\square$

Now the following proposition follows easily.

**Proposition 8.8 (Queue Equality).** *For all* $q, r : Queue$ *we have:*

$$q = r \ \leftrightarrow \ eq(q, r) = \mathsf{t}.$$

**Proof.** $\to$: By standard queue induction on $q$. $\leftarrow$: Easy, apply double queue induction as in the proof of Proposition 4.5, and use $d = e \leftrightarrow \overline{eq}(d, e) = \mathsf{t}$. $\square$

# 9 Tables

The tables are very similar to the queues, filled with pairs $\langle d, n\rangle$, where $d : D$, and $n : Nat$. We use $t_0$ to denote the empty table, and we omit the brackets from the pairs. There are two *test* functions on the tables: $test(n, t)$ says whether the table $t$ contains a pair $\langle d, n\rangle$, where $d$ is an arbitrary element from $D$. The boolean $test(d, n, t)$ says whether or not $t$ contains the pair $\langle d, n\rangle$.

**sort** $Table$
**cons** $t_0 :\to Table$
$in : D \times Nat \times Table \to Table$
**func** $rem : Nat \times Table \to Table$
$retr : Nat \times Table \to D$
$eq : Table \times Table \to \mathbf{Bool}$
$test : Nat \times Table \to \mathbf{Bool}$

| | | |
|---|---|---|
| | $test : D \times Nat \times Table \to \mathbf{Bool}$ | |
| | $size : Table \to Nat$ | |
| **var** | $m, n : Nat$ | |
| | $d, e : D$ | |
| | $t, u : Table$ | |
| **rew** | $rem(n, t_0) = t_0$ | Table1 |
| | $rem(m, in(d, n, t)) = if(eq(m, n), rem(m, t), in(d, n, rem(m, t)))$ | Table2 |
| | $retr(n, t_0) = d_0$ | Table3 |
| | $retr(m, in(d, n, t)) = if(eq(m, n), d, retr(m, t))$ | Table4 |
| | $eq(t_0, t_0) = \mathsf{t}$ | Table5 |
| | $eq(t_0, in(d, n, t)) = \mathsf{f}$ | Table6 |
| | $eq(in(d, n, t), t_0) = \mathsf{f}$ | Table7 |
| | $eq(in(d, m, t), in(e, n, u)) = and(and(\overline{eq}(d, e), eq(m, n)), eq(t, u))$ | Table8 |
| | $test(n, t_0) = \mathsf{f}$ | Table9 |
| | $test(m, in(d, n, t)) = or(eq(m, n), test(m, t))$ | Table10 |
| | $test(d, n, t_0) = \mathsf{f}$ | Table11 |
| | $test(d, m, in(e, n, t)) = or(and(eq(m, n), \overline{eq}(d, e)), test(d, m, t))$ | Table12 |
| | $size(t_0) = 0$ | Table13 |
| | $size(in(d, n, t)) = size(rem(n, t)) + 1$ | Table14 |

**Definition 9.1 (Standard Table Induction).** The following inference rule is referred to as *standard table induction on* $t : Table$:

$$\frac{\phi(t_0) \quad \phi(r) \to \phi(in(d, n, r))}{\phi(t)}$$

where $d : D$, $n : Nat$ and $r : Table$.

**Proposition 9.2.** *Proofs using standard table induction are correct.*

**Proof.** Follows from the above specification and the induction rule IND. □

We have some simple property formulas with tables.

**Proposition 9.3.** *For all* $t : Table$, $m, n : Nat$ *and* $d : D$ *we have:*

1. $test(n, rem(n, t)) = \mathsf{f}$,
2. $test(d, n, rem(n, t)) = \mathsf{f}$,
3. $eq(m, n) = \mathsf{f} \to test(m, rem(n, t)) = test(m, t)$,
4. $eq(m, n) = \mathsf{f} \to test(d, m, rem(n, t)) = test(d, m, t)$.

**Proof.** See Appendix. □

For tables we have that derivable equality is equivalent to equality as specified by the $eq$ function.

**Proposition 9.4 (Table Equality).** *For all* $t, u : Table$ *we have:*

$$t = u \ \leftrightarrow \ eq(t, u) = \mathsf{t}.$$

**Proof.** Completely analogous to the proof of Proposition 8.8. □

# 10 Windows

In the specification of *sliding window protocols* typical data structures can be used that are called *windows* (see e.g. [Tan89]). These windows are finite structures and usually defined using modulo arithmetic. In this section we study this data type more closely.

For the specification of windows, sets with natural numbers can be used, which are easily obtained from the specification in Section 6 when the sort name *Set* is replaced by *Nset*, and $D$ by *Nat*. The element $d_0$ is replaced by 0, and the empty set with naturals is defined by the constant $ns_0$. All equations and property formulas derived from the aforementioned specification can be straightforwardly translated to this new setting. We define one special function on the sort *Nset*: the function *window*. For $n > 1 = \mathsf{t}$ the function $window(k, m, n)$ denotes a set, filled with natural numbers from $k$ to $m$, modulo $n$. It is easy to verify, for instance, that $window(5, 2, 7)$ denotes the set $in(5, in(6, in(0, in(1, ns_0))))$.

| | | |
|---|---|---|
| **func** | $window : Nat \times Nat \times Nat \to Nset$ | |
| **var** | $k, m, n : Nat$ | |
| **rew** | $window(k, m, 0) = ns_0$ | Window1 |
| | $window(k, m, Sn) = if(eq(k_{Sn}, m_{Sn}), ns_0, in(k_{Sn}, window(k+1, m, Sn)))$ | Window2 |

The function *window* is defined on the equivalence classes induced by $.\,mod\, n$:

$$window(k_n, m, n) = window(k, m_n, n) = window(k, m, n).$$

We give the following property formulas for the *window* function.

**Proposition 10.1.** *For all $i, k, l, m, n : Nat$ we have:*

1. $window(k, m, 1) = ns_0$,
2. $window(m, m, n) = ns_0$,
3. $test(m_n, window(k, m, n)) = \mathsf{f}$,
4. $test(k_n, window(l, m, n)) = test(k +_n i, window(l + i, m + i, n))$,
5. $test(k_{Sn}, window(l, m, Sn)) =$
$or(and(l_{Sn} \leq k_{Sn}, k_{Sn} < m_{Sn}), or(and(m_{Sn} < l_{Sn}, l_{Sn} \leq k_{Sn}), and(k_{Sn} < m_{Sn}, m_{Sn} < l_{Sn})))$,
6. $eq(Sk_n, m_n) = \mathsf{f} \to and(not(eq(Sk_n, l_n)), test(Sk_n, window(l, m, n))) = test(k_n, window(l, m, n))$.

**Proof.** See Appendix. □

We give another three interesting property formulas.

**Proposition 10.2.** *For all $k, l, m, n : Nat$ we have:*

1. $eq(l_{Sn}, m_{Sn}) = \mathsf{f} \to not(test(k_{Sn}, window(l, m, Sn))) = test(k_{Sn}, window(m, l, Sn))$,
2. $k_n > m_n = \mathsf{t} \vee eq(k_n, 0) = \mathsf{t} \to window(k, m, n) = window(k, 0, n) \cup window(0, m, n)$,
3. $eq(k_{Sn}, Sm_{Sn}) = \mathsf{f} \to in(m_{Sn}, window(k, m, Sn)) = window(k, Sm, Sn)$.

**Proof.** For a proof of 10.2.1 identity 5 of the previous proposition can be used. The property formulas 10.2.2 and 10.2.3 can be proved by induction on $k$ using the schema for modulo induction in Definition 5.8.2. □

Now the sort *Nset* as well as the function *window* are defined it is not hard to specify boolean functions that can be used as predicates that indicate whether or not some natural number is in a given window. The function $\overline{inw}(k,l,m,n)$ is used to indicate whether $k_n$ is in the window, and $inw(k,l,m,n)$ is just as $\overline{inw}(k,l,m,n)$, but also equals $\mathsf{t}$ if $k_n = l_n$.

| | | |
|---|---|---|
| **func** | $inw, \overline{inw} : Nat \times Nat \times Nat \times Nat \to \mathbf{Bool}$ | |
| **var** | $k, l, m, n : Nat$ | |
| **rew** | $inw(k,l,m,0) = \mathsf{f}$ | Window3 |
| | $\overline{inw}(k,l,m,0) = \mathsf{f}$ | Window4 |
| | $inw(k,l,m,Sn) = test(k_{Sn}, window(l,m,Sn))$ | Window5 |
| | $\overline{inw}(k,l,m,Sn) = and(inw(k,l,m,Sn), not(eq(k_{Sn}, l_{Sn})))$ | Window6 |

We have the following basic facts about windows.

**Proposition 10.3.** *For all $i,k,l,m,n : Nat$ we have:*

1. $inw(k,l,m,1) = \mathsf{f}$,
2. $\overline{inw}(k,l,m,1) = \mathsf{f}$,
3. $\overline{inw}(k,l,m,n) = \overline{inw}(m,k,l,n)$,
4. $eq(Sk_n, m_n) = \mathsf{f} \to \overline{inw}(Sk,l,m,n) = inw(k,l,m,n)$,
5. $\overline{inw}(i,k,l,n) = \mathsf{t} \wedge \overline{inw}(i,m,k,n) = \mathsf{t} \to \overline{inw}(i,m,l,n) = \mathsf{t}$.

**Proof.** Follows quite easily from the specification and Proposition 10.1. □

# 11 Term rewriting properties

In this section we treat the term rewriting properties of the *term rewriting systems* (TRS's) associated with the specifications in this paper very briefly. Term rewriting is a well-known approach to formally proving properties of algebraic specifications (see e.g. [Klo92]).

Intuitively, a TRS can be regarded as a set of equations that can be applied only in one direction. Usually, the TRS associated with an algebraic specification is obtained by replacing the equality symbol (=) by an arrow (→), which indicates the rewrite direction, in all the equations. Thus equations become *rewrite rules*. A *rewrite step* denotes a single application of a rewrite rule. The 'nicest' property a TRS can have is that it is *complete*, which means that it is *strongly terminating*, i.e. allows no infinite sequences of rewrite steps, and that it generates *unique normal forms*. The first property guarantees that any automatic rewrite is a process that always terminates, and the second property guarantees that any *rewrite strategy* yields the same expression.

With all specifications, except those of the bags and the sets, complete term rewrite systems are associated. This can be easily seen, since the TRS's are *regular* (proving regularity is a standard technique for deciding completeness of TRS's). The bags and the sets form an exception, since the associated TRS's allow infinite sequences of rewrite steps, due to the axioms Bag7 and Set10 respectively. [2] We remark that the specification of sets also leads to a *non-linear* rewrite rule, since the variable $d$ occurs twice in the l.h.s. of axiom Set9.

W.r.t. the problem of our 'non-terminating' rules there sometimes is a simple solution. If the set $D$ is ordered according to some linear ordering $< : D \times D \to \mathbf{Bool}$, it is possible specify the bags in a different but equivalent way with the following *conditional* alternative for axiom Bag7 (which is not correct $\mu$CRL):

[2] A well-known technique for handling such 'non-terminating' rules is to consider a TRS *modulo* these rules.

**rew** $\quad in(d, in(e, b)) = in(e, in(d, b)) \;\; when \;\; d < e$ Bag7'

It is easy to see that this axiom cannot lead to a rule that can be applied infinitely often in succession. The ordering $<$ on $D$ is linear, so for all $d, e : D$ it satisfies $d < e = \mathsf{t} \rightarrow or(e < d, \overline{eq}(d, e)) = \mathsf{f}$. Now if $\overline{eq}(d, e) = \mathsf{f}$ then $or(d < e, e < d) = \mathsf{t}$, so with axiom Bag7' we find $in(d, in(e, b)) = in(e, in(d, b))$, and this property holds trivially for $\overline{eq}(d, e) = \mathsf{t}$ (since it then follows that $d = e$, see Section 6). As a consequence, all property formulas derivable for the bags can also be derived in a setting with axiom Bag7' instead of Bag7. An adapted specification for the sets can be given in a similar way. We conjecture that with these alternative specifications complete TRS's are associated.

## 12 Consistency of specifications

We briefly treat the issue of consistency in the context of $\mu$CRL. We give no explicit proofs for consistency of the data specifications in this paper.

**Definition 12.1.** Let $S \in \mathcal{S}$. A specification $((\mathcal{S}, F), E)$ is called *consistent* if and only if for some terms $s, t : S$ it is not possible to derive both $s = t$ and $\neg(s = t)$ from the axioms in $E$, using the proof rules of $\mu$CRL. □

It follows that inconsistency is equivalent to the derivability of $\mathsf{t} = \mathsf{f}$. Clearly, an inconsistent specification is useless; anything can be derived from it.

By definition of constructor sets, all closed terms of some sort are provably equal to terms built from constructors and possibly terms of other sorts. Now if it can be proved that two such 'constructor terms', syntactically different and not equal according to some $\overline{eq}$ function, are derivably equal, then the specification considered is inconsistent. A particular reason for using 'constructor terms' for consistency checks is that in term rewriting analysis these terms are usually the normal forms.

**Example 12.2.**] The natural numbers specified in this paper are consistent if and only if it cannot be proved that $0 = Sn$. This can be easily verified by inspecting the $eq$ function for natural numbers. The total set of equations that has to be considered is

$$\begin{aligned} E_{Nat} \;\; = \;\; & \{\text{Nat1}, \ldots, \text{Nat8}\} \cup \{\text{Mod1}, \ldots, \text{Mod4}\} \cup \{\text{Queue22}, \text{Queue23}\} \cup \\ & \{\text{Table13}, \text{Table14}\} \cup \{\text{Bag14}, \text{Bag15}\} \cup \{\text{Set21}, \text{Set22}\}. \end{aligned}$$

Note that for such a proof we need consistent specifications for the sorts **Bool**, *Queue*, *Table*, *Bag*, and *Set*.

To *actually* prove that $0 = Sn$ cannot be derived from $E_{Nat}$, term rewriting techniques can be used. It has to be shown that from the (complete) TRS associated with $E_{Nat}$ no reduction $Sn \twoheadrightarrow 0$ is possible ( $\twoheadrightarrow$ denotes the reflexive and transitive closure of the rewrite relation $\rightarrow$). □

## 13 Concluding remarks

We specified a number of standard data types and functions on it in the specification language $\mu$CRL, and we gave a large number of properties of these data types. Proofs were given within the formal framework of the proof theory of $\mu$CRL.

Some typical obstacles that were met in previous work with data specifications in $\mu$CRL have been taken away. In particular, we specified $eq$ functions with 'nice' computational properties, and we argued that non-trivial proofs can become quite easy once alternative induction schemata are available. Of course, correctness proofs for such schemata add some 'meta-theoretical' aspects to the proof theory of $\mu$CRL, but we think that the proof theory remains well-organised if some general propositions are added for standard data types.

It should be noted that w.r.t. the proof theory, consistency of specifications is also a 'meta-theoretical' notion to which no explicit attention was paid in previous work on $\mu$CRL. We argued that the idea behind a consistency check is simple. Consistency can be verified by showing that two syntactically different 'constructor terms' are not provably equal, provided that an $\overline{eq}$ function is available according to which these terms are not equal.

In general, we think that the analysis of data specifications in $\mu$CRL will benefit from using term rewriting techniques where some basic issues are involved: 1) for proving that a given set of functions is a constructor set, and 2) for proving that a specification is consistent.

This study is primarily meant for practical use and we hope that its merits will be twofold: in the first place that we provided sufficiently 'processed' material for a standard library of data types for $\mu$CRL and the Coq proof checker, and in the second place that the introduction and systematic study of new data types in $\mu$CRL will be less problematic than it was before. We are quite confident that our approach to proving properties of algebraic specifications within the formal framework of $\mu$CRL will easily carry over to related formalisms based on the theory of algebraic data types.

# References


[BG93] M. Bezem and J.F. Groote. A formal verification of the alternating bit protocol in the calculus of constructions. Technical Report Logic Group Preprint Series No. 88, Utrecht University, 1993.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[BT82] J.A. Bergstra and J.V. Tucker. The completeness of the algebraic specification methods for computable data types. *Information and Control*, 12:186–200, 1982.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[Dal83] D. van Dalen. *Logic and Structure*. Springer-Verlag, 1983.

[Dow93] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The Coq proof assistant user's guide. Version 5.8. Technical Report, INRIA – Rocquencourt, May 1993.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications I*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[Gog80] J.A. Goguen. How to prove algebraic inductive hypotheses without induction: with applications to the correctness of data type representations. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5th Conference on Automated Deduction*, pages 356–373. Springer-Verlag, 1980.

[GP90] J.F. Groote and A. Ponse. The syntax and semantics of $\mu$CRL. Technical Report CS-R9076, CWI, Amsterdam, 1990.

[GP91] J.F. Groote and A. Ponse. Proof theory for $\mu$CRL. Technical Report CS-R9138, CWI, Amsterdam, 1991.

[GP94] J.F. Groote and A. Ponse. Proof theory for $\mu$CRL: a language for processes with data. In D.J. Andrews, J.F. Groote and C.A. Middelburg, editors, *Proceedings of the International Workshop on Semantics of Specification Languages*, pages 232–251. Workshops in Computer Science, Springer-Verlag, 1994.

[GvdP93] J.F. Groote and J.C. van de Pol. A bounded retransmission protocol for large data packets. Technical Report Logic Group Preprint Series No. 100, Utrecht University, 1993.

[HH82] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239–266, 1982.

[KS93] H.P. Korver and J. Springintveld. A computer-checked verification of Milner's scheduler. Technical Report CS-R9371, CWI, Amsterdam, 1993.

[Kam93] G. Kamsteeg. A formal verification of the alternating bit protocol in $\mu$CRL. Technical Report 93-37, Leiden University, 1993.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume II*, pages 1–116. Clarendon, 1992.

[MG85] J. Meseguer and J.A. Goguen. Initiality, induction and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge University Press, 1985.

[TvD88] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, An Introduction (vol I)*. North-Holland, 1988.

[Tan89] A.S. Tanenbaum. *Computer Networks.* Prentice Hall, 1989.

[Wie90] R.J. Wieringa. *Algebraic Foundations for Dynamic Conceptual Models.* PhD thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1990.

[Wir90] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (vol II)*, pages 677–788. Elsevier Science Publishers B.V., 1990.

**Proposition 4.5 (Equality).** *For all $m, n : Nat$ we have:*

$$m = n \leftrightarrow eq(m,n) = \mathsf{t}.$$

**Proof.** Let $k : Nat$.
$\rightarrow$: By standard induction on $m$. If $m \equiv 0$ the result follows trivially. If $m \equiv Sk$ and $m = n$ then $eq(m,n) = eq(Sk, Sk)$, which equals $\mathsf{t}$ according to axiom Nat12 and the i.h.
$\leftarrow$: We first prove an intermediate result:

$$(i) \quad eq(m,0) = \mathsf{t} \rightarrow m = 0.$$

If $m \equiv 0$ then $(i)$ follows trivially. If $m \equiv Sk$ then using the axioms Nat11 and Bool1 we derive a contradiction from the assumption $eq(Sk,0) = \mathsf{t}$. From $\mathcal{F}$ we conclude $Sk = 0$, so $eq(Sk,0) = \mathsf{t} \rightarrow Sk = 0$, and $(eq(k,0) = \mathsf{t} \rightarrow k = 0) \rightarrow (eq(Sk,0) = \mathsf{t} \rightarrow Sk = 0)$. Using standard induction we conclude $(i)$.

Let $\phi(m,n) \equiv (eq(m,n) = \mathsf{t} \rightarrow m = n)$. We just proved $\phi(m,0)$, and $\phi(0,n)$ can be proved in a similar way. Our induction hypothesis is $\phi(k,l)$, where $k,l : Nat$. If $eq(Sk,Sl) = \mathsf{t}$ it follows from axiom Nat12 that $eq(k,l) = \mathsf{t}$. From the i.h. it follows now that $k = l$, so we also have $Sk = Sl$. We conclude that it follows from the i.h. that $eq(Sk,Sl) = \mathsf{t} \rightarrow Sk = Sl$, which is exactly $\phi(Sk,Sl)$. Now all the clauses of the double induction schema are proved we may conclude $\phi(m,n)$. $\square$

**Definition 5.2 (Special Induction 1).** Let $n : Nat$ with $n > 0 = \mathsf{t}$. The following inference rule is referred to as *special induction 1 on $p : Nat$*:

$$\frac{k < n = \mathsf{t} \rightarrow \phi(k) \quad \phi(m) \rightarrow \phi(m+n)}{\phi(p)}$$

where $k, m : Nat$.

**Proposition 5.2.** *Proofs using special induction 1 are correct.*

**Proof.** We prove that this special induction schema is equivalent to the standard induction schema in Definition 4.1, i.e. that

$$\phi(0) \wedge (\phi(l) \rightarrow \phi(Sl)) \longleftrightarrow (k < n = \mathsf{t} \rightarrow \phi(k)) \wedge (\phi(m) \rightarrow \phi(m+n)),$$

where $k, l, m, n : Nat$, and $n > 0 = \mathsf{t}$.
$\rightarrow$: It follows directly from $\phi(0) \wedge (\phi(l) \rightarrow \phi(Sl))$ that $\phi(k)$, so we also have $k < n = \mathsf{t} \rightarrow \phi(k)$. It also follows immediately that $\phi(m+n)$, so we also have $\phi(m) \rightarrow \phi(m+n)$.
$\leftarrow$: We do this part of the proof in two steps.
Step 1. Let $\Phi(k,l,n) \equiv (l * n \leq k = \mathsf{t} \wedge k < Sl * n = \mathsf{t} \rightarrow \phi(k))$. We prove $\Phi(k,l,n)$ by standard induction on $l$: $\Phi(k,0,n)$ follows directly from the premise of the special induction schema, so we show the induction step. Our i.h. is $\Phi(k,l,n)$. From $Sl * n \leq k + n = \mathsf{t} \wedge k + n < SSl * n = \mathsf{t}$ and the i.h. we can derive $\phi(k)$, and from $\phi(k)$ and the premise of the special induction schema we conclude $\phi(k+n)$, so $\Phi(k+n, Sl, n)$ holds. Now it is easy to derive $\Phi(k, Sl, n)$, so we may conclude $\Phi(k,l,n)$.
Step 2. Let $\Psi(l,k,n) \equiv (k < Sl * n = \mathsf{t} \rightarrow \phi(k))$. We prove $\Psi(l,k,n)$ prove by standard induction on $l$. We have that $\Psi(0,k,n) \leftrightarrow \Phi(k,0,n)$, so we only have to prove the induction step. If $k < SSl * n = \mathsf{t}$ then $k < Sl * n = \mathsf{t}$ or $Sl * n \leq k = \mathsf{t} \wedge k < SSl * n = \mathsf{t}$. We can combine the former case with the i.h. $\Psi(l,k,n)$ and the latter with $\Phi(k, Sl, n)$. In both cases we find $\phi(k)$, so $\Psi(l,k,n) \rightarrow \Psi(Sl,k,n)$, and we conclude $\Psi(l,k,n)$. Now we can substitute $k$ for $l$ in $\Psi(l,k,n)$, so $\phi(k)$ (remember $n > 0 = \mathsf{t}$). Finally, the premise of the standard induction schema follows trivially from $\phi(k)$. $\square$

**Proof of Proposition 5.5.** Property formulas 1 – 3 are proved by standard induction on $m$, and 4 – 7 are proved by induction on $m$ with the help of the special induction schema in Definition 5.1. We prove 4:

4. $(k + (m \, mod \, n)) \, mod \, n = (k + m) \, mod \, n.$

For $n = 0$ the result follows trivially from axiom Mod1, so we assume that $n > 0 = \mathsf{t}$. For $m < n = \mathsf{t}$ the result follows from axiom Mod2. Furthermore we derive

$$\begin{array}{rcl} (k + ((m + n) \, mod \, n)) \, mod \, n & \stackrel{5.5.3}{=} & (k + (m \, mod \, n)) \, mod \, n \\ & \stackrel{i.h.}{=} & (k + m) \, mod \, n \\ & \stackrel{5.5.3}{=} & ((k + m) + n) \, mod \, n \\ & \stackrel{4.6.2}{=} & (k + (m + n)) \, mod \, n, \end{array}$$

which completes the proof. □

**Proof of Proposition 5.6.** Property formulas 1 – 4 are easily proved, but 5 – 7 require some more effort. We sketch a proof of 5:

5. $k -_{Sn} (l +_{Sn} m) = (k -_{Sn} l) -_{Sn} m.$

A proof of 6 can be given along the same lines. Three intermediate results are needed (note that we can use Definition 5.3):

$$\begin{array}{ll} (i) & Sk -_n Sm = k -_n m, \\ (ii) & 0 -_{Sn} (m +_{Sn} 1) = (0 -_{Sn} m) -_{Sn} 1, \\ (iii) & k -_{Sn} (m +_{Sn} 1) = (k -_{Sn} m) -_{Sn} 1. \end{array}$$

Identity $(i)$ can be proved by distinguishing 4 cases: $k_n < n \dot{-} 1 = \mathsf{t} \wedge m_n < n \dot{-} 1 = \mathsf{t}$, $k_n < n \dot{-} 1 = \mathsf{t} \wedge m_n = n \dot{-} 1$, $k_n = n \dot{-} 1 \wedge m_n < n \dot{-} 1 = \mathsf{t}$ and $k_n = n \dot{-} 1 \wedge m_n = n \dot{-} 1$. Note that proving $(i)$ is trivial for $n = 0$.
Identity $(ii)$ can be proved by distinguishing 3 cases: $m_{Sn} = 0$, $0 < m_{Sn} = \mathsf{t} \wedge m_{Sn} < Sn \dot{-} 1 = \mathsf{t}$ and $m_{Sn} = Sn \dot{-} 1$.
Identity $(iii)$ can be proved by double induction. Let

$$\phi(k, m) \equiv k -_{Sn} (m +_{Sn} 1) = (k -_{Sn} m) -_{Sn} 1.$$

Then $\phi(k, 0)$ is easily proved, and $\phi(0, m)$ is exactly identity $(ii)$. We show the induction step:

$$\begin{array}{rcl} Sk -_{Sn} (Sm +_{Sn} 1) & = & Sk -_{Sn} SSm \\ & \stackrel{(i)}{=} & k -_{Sn} Sm \\ & = & k -_{Sn} (m +_{Sn} 1) \\ & \stackrel{i.h.}{=} & (k -_{Sn} m) -_{Sn} 1 \\ & \stackrel{(i)}{=} & (Sk -_{Sn} Sm) -_{Sn} 1. \end{array}$$

This finishes the proof of $(iii)$. Proposition 5.6.5 can now be proved by standard induction on $m$. For $m = 0$ the result follows easily. The induction step is as follows:

$$\begin{array}{rcl} k -_{Sn} (l +_{Sn} Sm) & = & k -_{Sn} (l +_{Sn} (m +_{Sn} 1)) \\ & \stackrel{5.6.4}{=} & k -_{Sn} ((l +_{Sn} m) +_{Sn} 1) \\ & \stackrel{(iii)}{=} & (k -_{Sn} (l +_{Sn} m)) -_{Sn} 1 \\ & \stackrel{i.h.}{=} & ((k -_{Sn} l) -_{Sn} m) -_{Sn} 1 \\ & \stackrel{(iii)}{=} & (k -_{Sn} l) -_{Sn} (m +_{Sn} 1) \\ & = & (k -_{Sn} l) -_{Sn} Sm. \end{array}$$

Property formula 7 is proved with the help of the following identity:

$$(iv) \quad k -_{Sn} (0 -_{Sn} m) = k +_{Sn} m.$$

Identity $(iv)$ can be proved by distinguishing 2 cases: $m_{Sn} = 0$ and $m_{Sn} > 0 = \mathsf{t}$.
We derive 7:

$$\begin{aligned} k -_{Sn} (l -_{Sn} m) &= k -_{Sn} ((l +_{Sn} 0) -_{Sn} m) \\ &\overset{5.6.6}{=} k -_{Sn} (l +_{Sn} (0 -_{Sn} m)) \\ &\overset{5.6.5}{=} (k -_{Sn} l) -_{Sn} (0 -_{Sn} m) \\ &\overset{(iv)}{=} (k -_{Sn} l) +_{Sn} m. \end{aligned}$$

□

**Proof of Proposition 5.7.** Most of the proofs are quite elaborate. Therefore we only give a sketch.

1. Easy.
2. Easy.
3. $eq(k +_n l, m +_n l) = eq(k_n, m_n)$. Some intermediate results are the following:

   $$\begin{array}{ll} (i) & m_{Sn} < Sn \dot{-} 1 = not(eq(Sm_{Sn}, 0)), \\ (ii) & eq(Sm_n, 0) = eq(S(m_n), n), \\ (iii) & eq(Sm_n, 0) = \mathsf{f} \to S(m_n) = Sm_n, \\ (iv) & eq(Sk_n, Sm_n) = eq(k_n, m_n), \end{array}$$

   Property formula $(ii)$ follows with the help of $(i)$, and $(iii)$ follows with $(ii)$. We only prove $(iv)$.
   For $n = 0$ $(iv)$ follows trivially, so we assume that $n > 0 = \mathsf{t}$.
   If $Sk_n = Sm_n = 0$ then according to $(ii)$ $S(k_n) = S(m_n) = n$, so

   $$eq(Sk_n, Sm_n) = eq(0,0) = eq(n,n) = eq(S(k_n), S(m_n)) = eq(k_n, m_n).$$

   If $Sk_n = 0$ and $eq(Sm_n, 0) = \mathsf{f}$ then according to $(ii)$ we have $S(k_n) = n$, and thus

   $$eq(Sk_n, Sm_n) = eq(0, Sm_n) \overset{(ii)}{=} eq(n, S(m_n)) = eq(S(k_n), S(m_n)) = eq(k_n, m_n).$$

   The case where $eq(Sk_n, 0) = \mathsf{f}$ and $Sm_n = 0$ is handled in the same way.
   If $eq(Sk_n, 0) = \mathsf{f}$ and $eq(Sm_n, 0) = \mathsf{f}$ then

   $$eq(Sk_n, Sm_n) \overset{(iii)}{=} eq(S(k_n), S(m_n)) = eq(k_n, m_n).$$

   The proof of 3 from $(iv)$ is simple now: apply standard induction on $l$.
4. Property formula 4 follows easily from 3:

   $$\begin{aligned} eq(k -_{Sn} l, m -_{Sn} l) &\overset{5.7.3}{=} eq((k -_{Sn} l) +_{Sn} l, (m -_{Sn} l) +_{Sn} l) \\ &\overset{5.6.7}{=} eq(k -_{Sn} (l -_{Sn} l), m -_{Sn} (l -_{Sn} l)) \\ &= eq(k_{Sn}, m_{Sn}). \end{aligned}$$

5. By case distinction: $k_{Sn} = 0$ and $k_{Sn} > 0 = \mathsf{t}$.

6. Follows from 3 in a similar way as 4.
7. By case distinction: $k_{Sn} + m_{Sn} < Sn = \mathsf{t}$ and $k_{Sn} + m_{Sn} \geq Sn = \mathsf{t}$.
8. It is easy to derive that $k -_{Sn} m < Sn -_{Sn} m = k +_{Sn} (Sn -_{Sn} m) < Sn -_{Sn} m$. Application of 7 yields the desired result.
9. By distinction of two cases: $k_{Sn} < l_{Sn} = \mathsf{t}$ and $k_{Sn} \geq l_{Sn} = \mathsf{t}$.

□

**Definition 5.8 (Modulo Induction).** Let $q : Nat$, and $n : Nat$ with $n > 0 = \mathsf{t}$. The following inference rules are referred to as *modulo induction on* $p : Nat$:

1. $$\frac{\phi(m_n, m_n) \quad \phi(k_n, m_n) \to \phi(Sk_n, m_n)}{\phi(p_n, q_n)}$$ *or*
2. $$\frac{\phi(m_n, m_n) \quad \phi(Sk_n, m_n) \to \phi(k_n, m_n)}{\phi(p_n, q_n)}$$

where $k, m : Nat$.

**Proposition 5.9.** *Proofs using modulo induction are correct.*

**Proof.** We prove that both schemata are equivalent to the standard schema. First we state two intermediate results: If $n > 0 = \mathsf{t}$ then

$$\begin{array}{ll} (i) & (\phi(k_n, m_n) \to \phi(Sk_n, m_n)) \to (\phi(k_n, m_n) \to \phi(k +_n l, m_n)), \\ (ii) & (\phi(Sk_n, m_n) \to \phi(k_n, m_n)) \to (\phi(Sk_n, m_n) \to \phi(k -_n l, m_n)), \end{array}$$

which can both easily be proved by standard induction on $l$.

1. $\to$: 1 follows nearly directly from the standard induction schema for natural numbers.

   $\leftarrow$: We have $\phi(k_n, m_n) \to \phi(Sk_n, m_n)$, so we only have to prove $\phi(0, m_n)$. From $(i)$ we may directly conclude $\phi(k_n, m_n) \to \phi(k +_n (n -_n k), m_n)$, and if we rewrite this result we find $\phi(k_n, m_n) \to \phi(0, m_n)$. Since we also have $\phi(m_n, m_n)$ we may conclude $\phi(0, m_n)$.

2. $\to$: From the standard induction schema we conclude $\phi(Sl_n, m_n) \to \phi(SSl_n, m_n)$, and from $(i)$ we conclude $\phi(Sl_n, m_n) \to \phi(Sl +_n (n -_n 1), m_n)$. Some simple calculations yield the desired result $\phi(Sl_n, m_n) \to \phi(l_n, m_n)$, and $\phi(m_n, m_n)$ follows trivially from the standard induction schema.

   $\leftarrow$: We have $\phi(Sk_n, m_n) \to \phi(k_n, m_n)$, so according to $(ii)$ we have $\phi(Sk_n, m_n) \to \phi(k -_n (n -_n 2), m_n)$. If we rewrite this last property formula we find $\phi(Sk_n, m_n) \to \phi(SSk_n, m_n)$. If we substitute $l -_n 1$ for $k$ we easily derive $\phi(l_n, m_n) \to \phi(Sl_n, m_n)$. It also follows from $(ii)$ that $\phi(Sk_n, m_n) \to \phi(k -_n k, m_n)$, which equals $\phi(Sk_n, m_n) \to \phi(0, m_n)$. Finally, if we substitute $m -_n 1$ for $k$, we conclude using $\phi(m_n, m_n)$ that $\phi(0, m_n)$ holds.

□

**Definition 6.5 (Special Set Induction 1).** The following inference rule is referred to as *special special set induction 1 on* $s : Set$:

$$\frac{\phi(s_0) \quad \phi(rem(d,t)) \to \phi(in(d,t))}{\phi(s)}$$

where $d : D$ and $t : Set$.

**Proposition 6.6.** *Proofs by special set induction with the above schema are correct.*

**Proof.** We prove that this special induction schema is equivalent to the standard induction schema in Definition 6.1, i.e. that

$$\phi(s_0) \wedge (\phi(s) \rightarrow \phi(in(d,s))) \longleftrightarrow \phi(s_0) \wedge (\phi(rem(d,t)) \rightarrow \phi(in(d,t))).$$

$\rightarrow$: Take $s = rem(d,t)$ and apply Proposition 6.3.1.
$\leftarrow$: If $test(d,t) = \mathsf{f}$ then according to Proposition 6.3.5 $rem(d,t) = t$, so the premise of special schema implies the premise of the standard schema. If $test(d,t) = \mathtt{t}$ then with Proposition 6.3.4 we find $\phi(t) \leftrightarrow \phi(in(d,t))$, so $\phi(s) \rightarrow \phi(in(d,s))$ follows trivially. $\square$

**Definition 6.7 (Special Set Induction 2).** Let $f : D \times Set \rightarrow Set$ be a function. The following inference rule is referred to as *special special set induction 2 on* $v : Set$:

$$\frac{\phi(s_0,t) \quad \phi(rem(d,s), f(d,t)) \rightarrow \phi(in(d,s),t)}{\phi(v,w)}$$

where $d : D$ and $s, t, w : Set$.

**Proposition 6.8.** *Proofs by special set induction with the above schema are correct.*

**Proof.** We prove that for all terms $v_0, w_0 : Set$ the property formula $\phi(v_0, w_0)$ follows from the premise of the second special induction schema, so we assume that the following holds:

$$\begin{array}{ll} (i) & \phi(s_0,t), \\ (ii) & \phi(rem(d,s), f(d,t)) \rightarrow \phi(in(d,s),t). \end{array}$$

We define an auxiliary function $g : Set \rightarrow Set$:

$$\begin{array}{ll} (iii) & g(s_0) \stackrel{\text{def}}{=} w_0, \\ (iv) & g(in(d,u)) \stackrel{\text{def}}{=} f(d, g(u)), \end{array}$$

where $u : Set$.

The following property formulas can be derived easily:

$$\begin{array}{ll} (v) & in(d,s) = v_0 \setminus u \;\rightarrow\; rem(d,s) = rem(d,v_0) \setminus rem(d,u), \\ (vi) & rem(d,v_0) \setminus rem(d, in(d,u)) = v_0 \setminus in(d,u). \end{array}$$

From $(ii)$ and $(v)$ it follows that

$$(vii) \quad in(d,s) = v_0 \setminus u \;\rightarrow\; (\phi(rem(d,v_0) \setminus rem(d,u), f(d,g(u))) \rightarrow \phi(v_0 \setminus u, g(u))).$$

Let $\Phi(d,u) \equiv (\phi(rem(d,v_0) \setminus rem(d, in(d,u)), g(in(d,u))) \rightarrow \phi(v_0 \setminus u, g(u)))$. With axiom Set2 and $(iv)$ it follows from $(vii)$ that $in(d,s) = v_0 \setminus u \;\rightarrow\; \Phi(d,u)$. From $(i)$ it follows that $\phi(s_0, g(u))$, so $s_0 = v_0 \setminus u \;\rightarrow\; \Phi(d,u)$. By standard set induction we have $s = v_0 \setminus u \;\rightarrow\; \Phi(d,u)$, so we may conclude $\Phi(d,u)$. Now with $(vi)$ we derive from $\Phi(d,u)$ that

$$(viii) \quad \phi(v_0 \setminus in(d,u), g(in(d,u))) \rightarrow \phi(v_0 \setminus u, g(u)).$$

Let $\Psi(t) \equiv (\phi(v_0 \setminus t, g(t)) \rightarrow \phi(v_0, w_0))$. It follows from $(viii)$ that $\phi(v_0 \setminus in(d,u), g(in(d,u))) \rightarrow \phi(v_0 \setminus s_0, g(s_0))$, so using $(iii)$ and axiom Set7 we conclude $\Psi(in(d,u))$. With this result we also find $\Psi(u) \rightarrow \Psi(in(d,u))$. Moreover it is easy to derive $\phi(v_0 \setminus s_0, g(s_0)) \rightarrow \phi(v_0, w_0)$, so we also have $\Psi(s_0)$. Using standard set induction we conclude $\Psi(t)$, and consequently $\Psi(v_0)$. Finally we have $v_0 \setminus v_0 = s_0$, so with $(i)$ we conclude from $\Psi(v_0)$ that $\phi(v_0, w_0)$. $\square$

**Proposition 6.9 (Set Equality).** *For all $s, t : Set$ we have:*

$$s = t \ \leftrightarrow \ eq(s,t) = \mathsf{t}.$$

**Proof.** By induction on $s$. Let $d : D$ and $u : Set$.
$\rightarrow$: We use the first schema for special induction. If $s \equiv s_0$ the result follows easily, and if $s \equiv in(d,u)$ and $s = t$ then $eq(s,t) = eq(in(d,u), in(d,u))$. Now we can first apply the axioms Set13 and Set20, and subsequently the i.h.
$\leftarrow$: This time we use the second schema for special induction. Let $\phi(s,t) \equiv (eq(s,t) = \mathsf{t} \rightarrow s = t)$. If $s \equiv s_0$ then $\phi(s,t)$ follows easily by standard induction on $t$. If $s \equiv in(d,u)$ then our i.h. is $\phi(rem(d,u), rem(d,t))$. Two cases are distinguished:
<u>Case 1.</u> If $test(d,t) = \mathsf{t}$ then it follows from $eq(in(d,u),t) = \mathsf{t}$ after application of axiom Set13 that $eq(rem(d,u), rem(d,t)) = \mathsf{t}$. From the i.h. we conclude $rem(d,u) = rem(d,t)$, as well as $in(d, rem(d,u)) = in(d, rem(d,t))$. According to Proposition 6.3.1 we also have $in(d,u) = in(d,t)$. Since $test(d,t) = \mathsf{t}$ we can apply Proposition 6.3.4, and we conclude that $in(d,t) = t$. Combining these results yields

$$(i) \quad test(d,t) = \mathsf{t} \rightarrow (\phi(rem(d,u), rem(d,t)) \rightarrow \phi(in(d,u),t)).$$

<u>Case 2.</u> If $test(d,t) = \mathsf{f}$ then according to axiom Set13 $eq(in(d,u),t) = \mathsf{f}$. So we can easily derive a contradiction. It follows that

$$(ii) \quad test(d,t) = \mathsf{f} \rightarrow (\phi(rem(d,u), rem(d,t)) \rightarrow \phi(in(d,u),t)).$$

Finally we combine $(i)$ and $(ii)$ with $\phi(s_0,t)$, and we conclude $\phi(s,t)$. $\square$

**Proof of Proposition 6.10.** By induction on $s$. Let $d : D$. Without proof we state the following intermediate results:

$$\begin{array}{ll} (i) & s_0 \subseteq s = \mathsf{t}, \\ (ii) & in(d,s) \subseteq in(d,t) = rem(d,s) \subseteq rem(d,t). \end{array}$$

1. $s \subset s = \mathsf{f}$. We use the first schema for special induction. If $s \equiv s_0$ then axiom Set14 can be applied. If $s \equiv in(d,t)$ then with the axioms Set16 and Set20 we find that $in(d,t) \subset in(d,t) = rem(d,t) \subset rem(d,t)$. It follows immediately from the i.h. that $in(d,t) \subset in(d,t) = \mathsf{f}$, so we conclude $s \subset s = \mathsf{f}$.

2. $s \cap t \subseteq t = \mathsf{t}$. We use the second schema for special induction.

   If $s \equiv s_0$ then $s \cap t \subseteq t = s_0 \cap t \subseteq t \stackrel{Set5}{=} s_0 \subseteq t \stackrel{(i)}{=} \mathsf{t}$.

   If $s \equiv in(d,u)$ then if $test(d,t) = \mathsf{t}$ then

$$\begin{array}{rcl} in(d,u) \cap t \subseteq t & \stackrel{Set6}{=} & in(d, u \cap t) \subseteq t \\ & \stackrel{6.3.4}{=} & in(d, u \cap t) \subseteq in(d,t) \\ & \stackrel{(ii)}{=} & rem(d, u \cap t) \subseteq rem(d,t) \\ & \stackrel{6.3.13}{=} & rem(d,u) \cap rem(d,t) \subseteq rem(d,t). \end{array}$$

   If $test(d,t) = \mathsf{f}$ then

$$\begin{array}{rcl} in(d,u) \cap t \subseteq t & \stackrel{6.3.1}{=} & in(d, rem(d,u)) \cap t \subseteq t \\ & \stackrel{Set6}{=} & rem(d,u) \cap t \subseteq t \\ & \stackrel{6.3.5}{=} & rem(d,u) \cap rem(d,t) \subseteq rem(d,t). \end{array}$$

   It follows from the i.h. that $in(d,u) \cap t \subseteq t = \mathsf{t}$, so we conclude $s \cap t \subseteq t = \mathsf{t}$.

3. $s \subseteq s \cup t = \mathsf{t}$. We use the second schema for special induction.

   If $s \equiv s_0$ then $s \subseteq s \cup t = s_0 \subseteq s_0 \cup t \overset{Set3}{=} s_0 \subseteq t \overset{(i)}{=} \mathsf{t}$.

   If $s \equiv in(d,u)$ then

$$\begin{aligned} in(d,u) \subseteq in(d,u) \cup t &\overset{Set4}{=} in(d,u) \subseteq in(d,u \cup t) \\ &\overset{(ii)}{=} rem(d,u) \subseteq rem(d,u \cup t) \\ &\overset{6.3.12}{=} rem(d,u) \subseteq rem(d,u) \cup rem(d,t) \\ &\overset{i.h.}{=} \mathsf{t}. \end{aligned}$$

The property formulas 4 – 8 follow in a similar fashion. □

**Definition 6.11 (Special Set Induction 3).** The following inference rule is referred to as *special special set induction 3 on* $s : Set$:

$$\frac{\phi(s_0) \quad test(d,t) = \mathsf{t} \wedge \phi(rem(d,t)) \rightarrow \phi(t)}{\phi(s)}$$

where $d : D$ and $t : Set$.

**Proposition 6.12.** *Proofs by special set induction with the above schema are correct.*

**Proof.** We assume that the premise of this induction schema is satisfied, so that we have

$(i)$ $\phi(s_0)$,
$(ii)$ $test(d,t) = \mathsf{t} \wedge \phi(rem(d,t)) \rightarrow \phi(t)$.

The following results are needed:

$(iii)$ $size(t) = 0 \rightarrow t = s_0$,
$(iv)$ $test(d,t) = \mathsf{t} \wedge size(t) = Sn \rightarrow size(rem(d,t)) = n$,

where $n : Nat$. Property formula $(iii)$ is easily proved by standard induction on $t$, and $(iv)$ follows with the help of Proposition 6.3.4: If $test(d,t) = \mathsf{t}$ then $Sn = size(t) = size(in(d,t))$, which equals $size(rem(d,t)) + 1$ by axiom Size2.

Let $\Phi(s,m) \equiv (size(s) = m \rightarrow \phi(s))$, where $m : Nat$. We prove $\Phi(s,m)$ by standard induction on $m$. $\Phi(s,0)$ follows easily from $(i)$ and $(iii)$. The i.h. is $\Phi(s,n)$. If $test(d,t) = \mathsf{t} \wedge size(t) = Sn$ then it follows from $(iv)$ and the i.h. that $\phi(rem(d,t))$. It follows from $(ii)$ that $\phi(t)$, so we have proved that $\Phi(s,n) \rightarrow (test(d,t) = \mathsf{t} \wedge size(t) = Sn \rightarrow \phi(t))$. It follows by standard set induction on $t$ that $\Phi(s,n) \rightarrow (size(t) = Sn \rightarrow \phi(t))$, and therefore that $\Phi(s,n) \rightarrow \Phi(s,Sn)$, and hence by induction we may conclude $\Phi(s,m)$. Finally we can substitute $size(s)$ for $m$, so that we find $\phi(s)$. □

**Proof of Proposition 7.3.** By induction on $b$. We prove 1:

1. $con(b,c) = con(c,b)$.

Two intermediate results are needed.

$(i)$ $con(c, b_0) = c$,
$(ii)$ $con(in(d,c),a) = con(c, in(d,a))$.

Identity $(i)$ is easily proved by induction on $c$. Identity $(ii)$ is also proved by induction on $c$: For $c \equiv b_0$ $(ii)$ follows easily, for $c \equiv in(e,b)$ we derive

$$\begin{aligned} con(in(d, in(e,b)), a) &\stackrel{Bag7}{=} con(in(e, in(d,b)), a) \\ &\stackrel{Bag4}{=} in(e, con(in(d,b), a)) \\ &\stackrel{i.h.}{=} in(e, con(b, in(d,a))) \\ &\stackrel{Bag4}{=} con(in(e,b), in(d,a)). \end{aligned}$$

Identity 7.3.1 is proved now by induction on $b$: For $b \equiv b_0$ we have

$$con(b_0, c) \stackrel{Bag3}{=} c \stackrel{(i)}{=} con(c, b_0),$$

and for $b \equiv in(d,a)$ we have

$$con(in(d,a),c) \stackrel{Bag4}{=} in(d, con(a,c)) \stackrel{i.h.}{=} in(d, con(c,a)) \stackrel{Bag4}{=} con(in(d,c),a) \stackrel{(ii)}{=} con(c, in(d,a)).$$

□

**Definition 7.4 (Special Bag Induction 1).** Let $f : D \times Bag \to Bag$ be a function. The following inference rule is referred to as *special bag induction 1 on* $v : Bag$:

$$\frac{\phi(b_0, c) \quad \phi(b, f(d,c)) \to \phi(in(d,b), c)}{\phi(v,w)}$$

where $d : D$ and $b, c, w : Bag$.

**Proposition 7.5.** *Proofs by special bag induction with the above schema are correct.*

**Proof.** We prove that for all terms $v_0, w_0 : Bag$ the property formula $\phi(v_0, w_0)$ follows from the premise of the special induction schema, so we assume that the following holds:

$(i)$ $\phi(b_0, c)$,
$(ii)$ $\phi(b, f(d,c)) \to \phi(in(d,b), c)$.

We define an auxiliary function $g : Bag \to Bag$:

$(iii)$ $g(b_0) \stackrel{\text{def}}{=} w_0$,
$(iv)$ $g(in(d,a)) \stackrel{\text{def}}{=} f(d, g(a))$,

where $a : Bag$.

The following property formulas for the *diff* function can be derived:

$(v)$ $in(d,b) = \mathit{diff}(v_0, a) \to b = \mathit{diff}(v_0, in(d,a))$,
$(vi)$ $\mathit{diff}(q,q) = q_0$.

Let $\Phi(d,a) \equiv (\phi(\mathit{diff}(v_0, in(d,a)), g(in(d,a)) \to \phi(\mathit{diff}(v_0,a), g(a)))$. With $(v)$ it follows from $(ii)$ that $in(d,b) = \mathit{diff}(v_0,a) \to \Phi(d,a)$. From $(i)$ it follows that $\phi(b_0, g(a))$, so we also have $b_0 = \mathit{diff}(v_0,a) \to \Phi(d,a)$. By standard induction we have $b = \mathit{diff}(v_0,a) \to \Phi(d,a)$, so we conclude $\Phi(d,a)$.

Let $\Psi(t) \equiv (\phi(\mathit{diff}(v_0,t), g(t)) \to \phi(v_0, w_0))$. It follows from $\Phi(d,a)$ that $\Psi(in(d,a))$, so we have $\Psi(a) \to \Psi(in(d,a))$. Moreover it is easy to derive $\phi(\mathit{diff}(v_0, b_0), g(b_0)) \to \phi(v_0, w_0)$, which is exactly $\Psi(b_0)$. Using standard induction it follows that $\Psi(t)$, and consequently $\Psi(v_0)$, so with $(vi)$ and $(i)$ we finally may conclude $\phi(v_0, w_0)$. □

**Proposition 7.6 (Bag Equality).** *For all $b, c : Bag$ we have:*

$$b = c \leftrightarrow eq(b,c) = \mathsf{t}.$$

**Proof.** $\to$: By standard bag induction on $b$. $\leftarrow$: By induction on $b$. We use the first schema for special bag induction. Let $\phi(b,c) \equiv (eq(b,c) = \mathsf{t} \to b = c)$. If $b \equiv b_0$ then $\phi(b,c)$ follows easily by standard induction on $c$. If $b \equiv in(d,a)$, with $a : Bag, d : D$, then our i.h. is $\phi(a, rem(d,c))$. Two cases are distinguished:
<u>Case 1.</u> If $test(d,c) = \mathsf{t}$ then it follows from Proposition 7.3.6 and the assumption $eq(in(d,a),c) = \mathsf{t}$ that we have $eq(in(d,a), in(d, rem(d,c))) = \mathsf{t}$. From this formula and axiom Bag10 it follows that $eq(a, rem(d,c)) = \mathsf{t}$, so using the i.h. we find $a = rem(d,c)$, and consequently $in(d,a) = in(d, rem(d,c))$, so using Proposition 7.3.6 again we conclude $in(d,a) = c$. Combining these results yields

$$(i)\ test(d,c) = \mathsf{t} \to (\phi(a, rem(d,c)) \to \phi(in(d,a),c)).$$

<u>Case 2.</u> If $test(d,c) = \mathsf{f}$ then it follows from axiom Bag10 that we have $eq(in(d,a),c) = \mathsf{f}$. Now it is easy to derive a contradiction, and it follows that

$$(ii)\ test(d,c) = \mathsf{f} \to (\phi(a, rem(d,c)) \to \phi(in(d,a),c)).$$

Finally we combine $(i)$ and $(ii)$ with $\phi(b_0, c)$, and we conclude $\phi(b,c)$. □

**Definition 8.4 (Special Queue Induction).** Let $f : Queue \to Queue$ be a function. The following inference rule is referred to as *special queue induction on $v : Queue$*:

$$\frac{\phi(q_0, r) \quad \phi(q, f(r)) \to \phi(in(d,q), r)}{\phi(v,w)}$$

where $d : D$ and $q, r, w : Queue$.

**Proposition 8.5.** *Proofs using special queue induction are correct.*

**Proof.** We prove that for all terms $v_0, w_0 : Queue$ the property formula $\phi(v_0, w_0)$ follows from the premise of the special induction schema, so we assume that the following holds:

$$\begin{array}{ll} (i) & \phi(q_0, r), \\ (ii) & \phi(q, f(r)) \to \phi(in(d,q), r). \end{array}$$

We define two auxiliary functions. Let $s : Queue$. The function $g : Queue \to Queue$ is defined by

$$\begin{array}{ll} (iii) & g(q_0) \stackrel{\text{def}}{=} w_0, \\ (iv) & g(in(d,s)) \stackrel{\text{def}}{=} f(g(s)), \end{array}$$

and $diff : Queue \times Queue \to Queue$ by

$$\begin{array}{ll} (v) & diff(q, q_0) \stackrel{\text{def}}{=} q, \\ (vi) & diff(q, in(d,r)) \stackrel{\text{def}}{=} tl(diff(q,r)). \end{array}$$

The following property formulas for the *diff* function can be derived:

$$\begin{array}{ll} (vii) & in(d,q) = diff(v_0, s) \to q = diff(v_0, in(d,s)), \\ (viii) & diff(q,q) = q_0. \end{array}$$

Let $\Phi(d,s) \equiv (\phi(diff(v_0, in(d,s)), g(in(d,s))) \to \phi(diff(v_0,s), g(s)))$. With $(vii)$ it follows from $(ii)$ that $in(d,q) = diff(v_0,s) \to \Phi(d,s)$. From $(i)$ it follows that $\phi(q_0, g(s))$, so we also have $q_0 = diff(v_0,s) \to \Phi(d,s)$. By standard induction we have $q = diff(v_0,s) \to \Phi(d,s)$, so we conclude $\Phi(d,s)$.

Let $\Psi(t) \equiv (\phi(\mathit{diff}(v_0,t),g(t)) \to \phi(v_0,w_0))$. It follows from $\Phi(d,s)$ that $\Psi(in(d,s))$, so we have $\Psi(s) \to \Psi(in(d,s))$. Moreover it is easy to derive $\phi(\mathit{diff}(v_0,q_0),g(q_0)) \to \phi(v_0,w_0)$, which is exactly $\Psi(q_0)$. Using standard induction it follows that $\Psi(t)$, and consequently $\Psi(v_0)$, so with $(viii)$ and $(i)$ we conclude $\phi(v_0,w_0)$. □

**Definition 8.6 (Double Queue Induction).** The following inference rule is referred to as *double queue induction on* $v,w$ : *Queue*:

$$\frac{\phi(q_0,r) \quad \phi(q,q_0) \quad \phi(q,r) \to \phi(in(d,q),in(e,r))}{\phi(v,w)}$$

where $d,e : D$ and $q,r : \mathit{Queue}$.

**Proposition 8.7.** *Proofs using double queue induction are correct.*

**Proof.** We prove that the property formula $\phi(v,w)$ follows from the premise of the double induction schema, so we assume that

$(i)$ $\phi(q_0,r)$,
$(ii)$ $\phi(q,q_0)$,
$(iii)$ $\phi(q,r) \to \phi(in(d,q),in(e,r))$.

Let $\Phi(d,q,t) \equiv (\phi(q,tl(t)) \to \phi(in(d,q),t))$, where $t : \mathit{Queue}$. If $t = in(i,s)$ for some $s : \mathit{Queue}, i : D$, then $tl(t) = s$, so using $(iii)$ we find that $t = in(i,s) \to \Phi(d,q,t)$, and using $(ii)$ we also find that $t = q_0 \to \Phi(d,q,t)$. We can now apply standard queue induction, so we have $t = s \to \Phi(d,q,t)$, and consequently $\Phi(d,q,t)$. If we combine this result with $\phi(q_0,t)$, which follows from $(i)$, we exactly have the premise of the special induction schema in Definition 8.4, so we conclude $\phi(v,w)$. □

**Proposition 9.3.** *For all* $t : \mathit{Table}$, $m,n : \mathit{Nat}$ *and* $d : D$ *we have:*

1. $test(n,rem(n,t)) = \mathsf{f}$,
2. $test(d,n,rem(n,t)) = \mathsf{f}$,
3. $eq(m,n) = \mathsf{f} \to test(m,rem(n,t)) = test(m,t)$,
4. $eq(m,n) = \mathsf{f} \to test(d,m,rem(n,t)) = test(d,m,t)$.

**Proof.** Let $s : \mathit{Table}$, $k : \mathit{Nat}$ and $e : D$.

1. By induction on $t$. For $t \equiv t_0$ we have $test(n,rem(n,t_0)) \overset{Table1}{=} test(n,t_0) \overset{Table9}{=} \mathsf{f}$, so identity 9.3.1 follows immediately. Now suppose that $t \equiv in(e,m,s)$, then
$$test(n,rem(n,in(e,m,s))) \overset{Table2}{=} test(n,if(eq(m,n),rem(n,s),in(e,m,rem(n,s)))).$$
If $eq(m,n) = \mathsf{t}$ then $test(n,rem(n,in(e,m,s))) = test(n,rem(n,s))$. If $eq(m,n) = \mathsf{f}$ then $test(n,rem(n,in(e,m,s))) = test(n,in(e,m,rem(n,s)))$. Using Table10 and $eq(m,n) = \mathsf{f}$, we see that the r.h.s. of this equation again equals $test(n,rem(n,s))$. Now we can apply the i.h. and we conclude that Proposition 9.3.1 holds.

2. Very similar to the previous proof.

3. By induction on $t$. For $t \equiv t_0$ property formula 9.3.3 follows immediately. Let $t \equiv in(e,k,s)$, then if $eq(m,n) = \mathsf{f}$ and $eq(k,n) = \mathsf{t}$, then

$$\begin{aligned} test(m, rem(n, in(e,k,s))) &\stackrel{Table2}{=} test(m, rem(n,s)) \\ &\stackrel{i.h.}{=} test(m,s) \\ &\stackrel{Table10}{=} test(m, in(e,k,s)), \end{aligned}$$

if $eq(k,m) = \mathsf{t}$, and $eq(k,n) = \mathsf{f}$, then

$$\begin{aligned} test(m, rem(n, in(e,k,s))) &\stackrel{Table2}{=} test(m, in(e,k,rem(n,s))) \\ &\stackrel{Table10}{=} \mathsf{t} \\ &\stackrel{Table10}{=} test(m, in(e,k,s)), \end{aligned}$$

and if $eq(m,n) = \mathsf{f}$, $eq(k,m) = \mathsf{f}$, and $eq(k,n) = \mathsf{f}$, then

$$\begin{aligned} test(m, rem(n, in(e,k,s))) &\stackrel{Table2}{=} test(m, in(e,k,rem(n,s))) \\ &\stackrel{Table10}{=} test(m, rem(n,s)) \\ &\stackrel{i.h.}{=} test(m,s) \\ &\stackrel{Table10}{=} test(m, in(e,k,s)). \end{aligned}$$

4. Very similar to the previous proof.

□

**Proposition 10.1.** *For all $i,k,l,m,n : Nat$ we have:*

1. $window(k,m,1) = ns_0$,
2. $window(m,m,n) = ns_0$,
3. $test(m_n, window(k,m,n)) = \mathsf{f}$,
4. $test(k_n, window(l,m,n)) = test(k +_n i, window(l+i, m+i, n))$,
5. $test(k_{Sn}, window(l,m,Sn)) =$
   $or(and(l_{Sn} \leq k_{Sn}, k_{Sn} < m_{Sn}), or(and(m_{Sn} < l_{Sn}, l_{Sn} \leq k_{Sn}), and(k_{Sn} < m_{Sn}, m_{Sn} < l_{Sn})))$,
6. $eq(Sk_n, m_n) = \mathsf{f} \rightarrow and(not(eq(Sk_n, l_n)), test(Sk_n, window(l,m,n))) = test(k_n, window(l,m,n))$.

**Proof.** Since some of the proofs are quite elaborate, we only give a sketch.

1. According to Proposition 5.5.2 we have $k_1 = m_1 = 0$, so 1 follows immediately from axiom Window2.
2. Directly from Window2.
3. For $n = 0$ property formula 3 follows trivially, so we assume that $n > 0 = \mathsf{t}$. We can use the induction schema in Definition 5.8.2. If $k_n = m_n$ then 3 follows from axiom Window2, and for $eq(k_n, m_n) = \mathsf{f}$ we derive

$$\begin{aligned} test(m_n, window(k,m,n)) &\stackrel{Window2}{=} test(m_n, in(k_n, window(Sk,m,n))) \\ &= test(m_n, window(Sk,m,n)) \\ &\stackrel{i.h.}{=} \mathsf{f}. \end{aligned}$$

4. We first give two intermediate results:

$$\begin{array}{ll} (i) & n > 0 = \mathsf{t} \to test(k_n, window(l,m,n)) = \\ & \qquad and(not(eq(l_n, m_n)), or(eq(k_n, l_n), test(k_n, window(Sl,m,n)))), \\ (ii) & test(k_n, window(l,m,n)) = test(Sk_n, window(Sl, Sm, n)). \end{array}$$

   Property formula $(i)$ follows from axiom Window2, and $(ii)$ can be proved by induction on $l$, using the induction schema in Definition 5.8.2. For $n = 0$ $(ii)$ follows trivially, so we assume that $n > 0 = \mathsf{t}$. If $l_n = m_n$ then $(ii)$ follows easily from axiom Window2, and in general we derive

$$\begin{aligned} & test(k_n, window(l,m,n)) \\ \stackrel{(i)}{=}\ & and(not(eq(l_n, m_n)), or(eq(k_n, l_n), test(k_n, window(Sl,m,n)))) \\ \stackrel{i.h.}{=}\ & and(not(eq(l_n, m_n)), or(eq(k_n, l_n), test(Sk_n, window(SSl,Sm,n)))) \\ \stackrel{5.7.3}{=}\ & and(not(eq(Sl_n, Sm_n)), or(eq(Sk_n, Sl_n), test(Sk_n, window(SSl,Sm,n)))) \\ \stackrel{(i)}{=}\ & test(Sk_n, window(Sl,Sm,n)). \end{aligned}$$

   Property formula 4 now follows easily from $(ii)$ by standard induction on $i$.

5. A proof can be given that uses the following intermediate results:

$$\begin{array}{ll} (iii) & test(k_{Sn}, window(0,m,Sn)) = k_{Sn} < m_{Sn}, \\ (iv) & test(k_{Sn}, window(m,0,Sn)) = and(k_{Sn} \geq m_{Sn}, m_{Sn} > 0), \\ (v) & l_{Sn} < m_{Sn} = \mathsf{t} \to test(k_{Sn}, window(l,m,Sn)) = and(l_{Sn} \leq k_{Sn}, k_{Sn} < m_{Sn}), \\ (vi) & l_{Sn} > m_{Sn} = \mathsf{t} \to test(k_{Sn}, window(l,m,Sn)) = or(l_{Sn} \leq k_{Sn}, k_{Sn} < m_{Sn}). \end{array}$$

   Here we need some of the results from Proposition 5.7. Identity $(iii)$ can be proved by double induction on $k, m$, and property formula $(iv)$ follows from 4, $(iii)$ and 5.7.8. Property formula $(v)$ follows from 4, $(iii)$ and 5.7.9, and $(vi)$ follows from 4, $(iv)$, and 5.7.9. We prove $(iv)$:

$$\begin{aligned} & test(k_{Sn}, window(m,0,Sn)) \\ \stackrel{(10.1.4)}{=}\ & test(k +_{Sn} (Sn \dot{-} m_{Sn}), window(m + (Sn \dot{-} m_{Sn}), 0 + (Sn \dot{-} m_{Sn}), Sn)) \\ =\ & test(k -_{Sn} m, window(0, Sn \dot{-} m_{Sn}, Sn)) \\ \stackrel{(iii)}{=}\ & k -_{Sn} m < Sn \dot{-} m_{Sn} \\ \stackrel{5.7.8}{=}\ & and(k_{Sn} \geq m_{Sn}, m_{Sn} > 0). \end{aligned}$$

   Finally, property formula 5 can be obtained by combining $(v), (vi)$ and 2.

6. Trivial for $n = 0$, for $n > 0 = \mathsf{t}$ property formula 6 can be proved by distinguishing the following two cases: $k_n = n \dot{-} 1$ and $k_n < n \dot{-} 1 = \mathsf{t}$ (in the former case it can be used that $Sk_n = 0$ and in the latter we have $Sk = S(k_n)$).

□