Inductive Proofs with Sets, and some Applications in Process Algebra

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

We formalise proofs by structural induction on sets, specified as an algebraic data type. The core idea is that the standard scheme for constructor induction can be used for the derivation of alternative schemes. This way, considerable flexibility can be obtained for proofs in the inductive theory for the sets. For more general purposes we formulate a rule for 'hybrid' induction, which allows a variety of induction schemes.

A number of examples is provided, most of which contain well-known and intuitively clear facts about sets that are nevertheless not always easy to prove. We also use our material on sets for proving some interesting properties of two special processes that have sets as a parameter, specified in μCRL style, representing generalised alternative and parallel compositions of processes. We moreover demonstrate how our generalised parallel composition can be used for a specification of broadcasting.

1991 Mathematics Subject Classification: 68Q40, 68Q60, 68Q65.
1987 CR Categories: D.3.1, D.3.3, F.4.1.
Key Words & Phrases: Abstract Data Types, Algebraic Specification, Inductive Proofs, Process Algebra, μCRL.

1 Introduction

The aim of this paper is to formalise proofs by structural induction on sets, specified as an algebraic data type. We hope to demonstrate that 'everything you always wanted to know about sets' can be be proved in an elegant, straightforward manner, once the right induction schemes are available. Another thing we wish to establish is that process algebras that use algebraic data types can benefit from knowledge that primarily concerns data. The specific formalism we use throughout this paper is the process specification language μCRL, and its proof theory [GP90, GP91]. For studying sets as a data type, however, we could have chosen any other formalism based on algebraic specification theory as well.

Data types in μCRL are algebraically specified in the standard way, using sorts, functions and axioms (see e.g. [MG85, Wir90]). Process specification in μCRL is based on the pro-

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

cess algebra ACP (Algebra of Communicating Processes, see e.g. [BK84, BW90]), which is particularly suited for the formal description of concurrent, communicating systems.

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. The proof theory of μCRL can be regarded as a logical framework, defined over a language of *property formulas*, which contains identities between data and identities between processes, linked with the propositional connectives $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$. The proof system also supports proofs by *constructor induction*, which is a well-known technique for proving properties of data by structural induction on terms, with an induction rule.

As a result of this purely algebraic character, verifications in μCRL lend themselves very well for automated proof checking. As examples of such work we mention [BG93, KS93]. For a brief overview of μCRL and its proof theory, and a survey of verifications of concurrent systems in it we refer to the paper [GP94]. The derivation and use of alternative induction schemes as we use it originates from [GvW94], where a large collection of standard data types and structures was investigated. This approach was in its turn inspired by [TvD88], where a scheme for *double induction* on natural numbers was derived.

Preliminaries. As to the property language of μCRL, we use the convention that $=$ binds stronger than any of the logical operators, that $\neg$ binds stronger than any of the binary logical operators, and that $\wedge, \vee$ bind stronger than $\rightarrow, \leftrightarrow$. Variables are implicitly universally quantified. For readability, we do not always strictly adhere to the syntax of μCRL.

In this paper we assume an algebraic specification of a sort **Bool**, with constants t and f, satisfying $\neg(\mathsf{t} = \mathsf{f})$. We also assume a unary function *not* and binary functions *and* and *or* with obvious meanings. Also a specification of a sort *Nat* is assumed, which consists of a constant 0, a unary function S (successor), and addition $+$.

As a consequence of the induction rule in μCRL, two very useful and common induction schemes can be formulated for these sorts. Let ϕ, ψ be property formulas, b : **Bool**, and $m, n : Nat$, then

$$\begin{array}{ll} \text{(Case Distinction)} & \phi(\mathsf{t}) \wedge \phi(\mathsf{f}) \rightarrow \phi(b), \\ \text{(Standard Induction)} & \psi(0) \wedge (\psi(m) \rightarrow \psi(Sm)) \rightarrow \psi(n). \end{array}$$

Both induction schemes are used in this paper, which is further organised as follows: In Section 2 an algebraic specification of sets is given. There we formulate the 'standard induction scheme' for sets, and we derive a number of induction schemes that can make life a lot easier. In Section 3 a rule for 'hybrid' induction on sets is provided, and in Section 4 we demonstrate how sets interact with processes by means of an example with generalised sums and merges. We also give an example which shows that a broadcast process can be modelled with a generalised merge.

Acknowledgements. I would like to thank Jan Bergstra, Willem Jan Fokkink Henri Korver and Alban Ponse for helpful comments. Special thanks go also to Henri for drawing my attention to the generalised merge, which was used in the specification and verification of Milner's famous scheduler [Mil89]. In his paper [KS93] he had a hard job handling this special process operator.

2 Algebraic specification of sets

Sets with elements of some arbitrary sort D are specified in a standard fashion. [1] We do not need any specific assumptions on D, except that there has to be a function $eq : D \times D \to \mathbf{Bool}$ satisfying $d = e \leftrightarrow eq(d,e) = \mathsf{t}$ for $d, e : D$. The empty set is represented by the constant s_0, and, not in accordance with the definition of μCRL, we use infix operators $\cup$, $\cap$, and $\setminus$ for representing the standard functions on sets.

sort	Set	
cons	$s_0 :\to Set$	
	$in : D \times Set \to Set$	
func	$rem : D \times Set \to Set$	
	$\cup, \cap, \setminus : Set \times Set \to Set$	
	$if : \mathbf{Bool} \times Set \times Set \to Set$	
	$test : D \times Set \to \mathbf{Bool}$	
var	$d, e : D$	
	$s, t : Set$	
rew	$if(\mathsf{t}, s, t) = s$	Con1
	$if(\mathsf{f}, s, t) = t$	Con2
	$test(d, s_0) = \mathsf{f}$	Test1
	$test(d, in(e,s)) = or(eq(d,e), test(d,s))$	Test2
	$in(d, in(d,s)) = in(d,s)$	Sets1
	$in(d, in(e,s)) = in(e, in(d,s))$	Sets2
	$rem(d, s_0) = s_0$	Sets3
	$rem(d, in(e,s)) = if(eq(d,e), rem(d,s), in(e, rem(d,s)))$	Sets4
	$s_0 \cup s = s$	Sets5
	$in(d,s) \cup t = in(d, s \cup t)$	Sets6
	$s_0 \cap s = s_0$	Sets7
	$in(d,s) \cap t = if(test(d,t), in(d, s \cap t), s \cap t)$	Sets8
	$s \setminus s_0 = s$	Sets9
	$s \setminus in(d,t) = rem(d, s \setminus t)$	Sets10

The characteristic properties of sets are captured by axiom Sets1, which implies that every element in a set occurs only once, and axiom Sets2, 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, and commutativity of the operators $\cup$ and $\cap$.

The standard scheme for constructor induction on sets is given below.

Definition 1 (Standard Set Induction). A proof of a property formula $\phi(s)$, by standard set induction on $s : Set$, is the deduction

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

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

[1] The presentation of the axioms, and therefore the naming, differs from [GvW94].

Proposition 2. *Proofs by standard set induction are correct.*

Proof A proof that anything derived with the above scheme is valid in the inductive theory, i.e., for all closed terms, is standard. First it has to be shown that the standard scheme follows from the induction rule of μCRL, parameterised with the set $\{s_0 :\to Set, in : D \times Set \to Set\}$, and then it has to be proved from the above specification that every closed term of sort Set can be represented as either s_0 or $in(d,t)$ for some $d : D$ and $t : Set$, i.e., that the section **cons** indeed specifies a *constructor set*. Finally the soundness of the above scheme follows from the soundness of the induction rule. □

We start off with some facts about the most basic operations on sets. These property formulas are used later on in proofs.

Proposition 3 (Simple Set Indentities). *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. $test(d, s \cup t) = or(test(d, s), test(d, t))$,
7. $test(d, s \cap t) = and(test(d, s), test(d, t))$,
8. $rem(d, s \cup t) = rem(d, s) \cup rem(d, t)$,
9. $rem(d, s \cap t) = rem(d, s) \cap rem(d, t)$,
10. $test(d, rem(e, s)) = \mathsf{t} \to in(e, rem(d, s)) = rem(d, in(e, s))$.

Proof. By induction on s. As an example we prove 1:
We have $in(d, rem(d, s_0)) \overset{Sets3}{=} in(d, s_0)$.
If $s \equiv in(e, t)$ then if $eq(d, e) = \mathsf{t}$

$$\begin{aligned} in(d, rem(d, in(e,t))) &\overset{Sets4,Con1}{=} in(d, rem(d,t)) \overset{i.h.}{=} in(d,t) \\ &\overset{Sets1}{=} in(d, in(d,t)) = in(d, in(e,t)). \end{aligned}$$

If $eq(d, e) = \mathsf{f}$ then

$$\begin{aligned} in(d, rem(d, in(e,t))) &\overset{Sets4,Con2}{=} in(d, in(e, rem(d,t))) \overset{Sets2}{=} in(e, in(d, rem(d,t))) \\ &\overset{i.h.}{=} in(e, in(d,t)) \overset{Sets2}{=} in(d, in(e,t)). \end{aligned}$$

□

Note the correspondence between the *rem* function (propositions 3.2 and 3.3), and the *in* function (axioms Sets1 and Sets2).

At this stage we can already formulate an alternative induction scheme, which is a variant of the standard one.

Definition 4 (Special Set Induction 1). A proof of a property formula $\phi(s)$, by special set induction 1 on $s : Set$, is the deduction

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

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

Proposition 5. *Proofs by special set induction with the above scheme are correct.*

Proof. We prove that this special induction scheme is equivalent to the standard induction scheme in Definition 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 3.1.
$\leftarrow$: If $test(d,t) = \mathsf{f}$ then according to Proposition 3.5 $rem(d,t) = t$, so the premise of special scheme implies the premise of the standard scheme. If $test(d,t) = \mathsf{t}$ then with Proposition 3.4 we find $\phi(t) \leftrightarrow \phi(in(d,t))$, so $\phi(s) \rightarrow \phi(in(d,s))$ follows trivially. $\square$

There are a number of well-known properties of operations on sets, such as commutativity, associativity, idempotency, absorption of $\cup$ and $\cap$, and distribution laws, such as

$$s \cap (t \cup u) = (s \cap t) \cup (s \cap u), \qquad (t \cap u) \setminus s = (t \setminus s) \cap (u \setminus s).$$

All these laws can be derived using the presented material and a number of other basic property formulas. Of course one also needs the scheme for standard induction on sets, as well as properties of the booleans.

The following step we take is the introduction of some familiar boolean functions on sets. We axiomatise equality of sets since it can be useful to have equality as a predicate *eq*, e.g. for use in the conditional construct *if*. Moreover we define subset predicates $\subset$ and $\subseteq$, which also uses *eq*.

func	$eq, \subset, \subseteq : Set \times Set \rightarrow \mathbf{Bool}$	
rew	$eq(s_0, s_0) = \mathsf{t}$	Eq1
	$eq(s_0, in(d,s)) = \mathsf{f}$	Eq2
	$eq(in(d,s), t) = and(test(d,t), eq(rem(d,s), rem(d,t)))$	Eq3
	$s_0 \subset s_0 = \mathsf{f}$	Sub1
	$s_0 \subset in(d,s) = \mathsf{t}$	Sub2
	$in(d,s) \subset t = and(test(d,t), rem(d,s) \subset rem(d,t))$	Sub3
	$s \subseteq t = or(eq(s,t), s \subset t)$	Sub4

For proving property formulas with sets that contain the above boolean functions, a useful new induction scheme can be introduced. The reason for this becomes clear when looking at the specification, and in particular at the axioms Eq3 and Sub3. E.g. the definition of $eq(in(d,s),t)$ contains $eq(rem(d,s), rem(d,t))$ in the r.h.s., so in order to prove a property formula $\phi(eq(v,w))$, it would be very helpful to have $\phi(eq(rem(d,s), rem(d,t)))$ as induction hypothesis. This is possible according to the following special induction scheme, which allows function applications on two variables. Note that axiom Sub3 has a similar inductive structure as axiom Eq3, and that axiom Sub4 will give no problems once the functions eq and $\subset$ are studied.

Definition 6 (Special Set Induction 2). Let $f : D \times Set \to Set$ be a function. A proof of a property formula $\phi(v,w)$, by special set induction 2 on $v : Set$, is the deduction

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

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

Proposition 7. *Proofs by special set induction with the above scheme 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 scheme, so we assume that the following holds:

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

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

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

where $u : Set$.

The following property formulas can be derived easily:

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

From (ii) and (v) it follows that

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

Let $\Phi(d,u) \equiv (\phi(rem(d,v_0) \setminus rem(d, in(d,u)), g(in(d,u))) \to \phi(v_0 \setminus u, g(u)))$. With axiom Sets4 and (iv) it follows from (vii) that $in(d,s) = v_0 \setminus u \ \to \ \Phi(d,u)$. From (i) it follows that $\phi(s_0, g(u))$, so $s_0 = v_0 \setminus u \ \to \ \Phi(d,u)$. By standard set induction we have $s = v_0 \setminus u \ \to \ \Phi(d,u)$, so we may conclude $\Phi(d,u)$. Now with (vi) we derive from $\Phi(d,u)$ that

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

Let $\Psi(t) \equiv (\phi(v_0 \setminus t, g(t)) \to \phi(v_0, w_0))$. It follows from $(viii)$ that $\phi(v_0 \setminus in(d,u), g(in(d,u))) \to \phi(v_0 \setminus s_0, g(s_0))$, so using (iii) and axiom Sets9 we conclude $\Psi(in(d,u))$. With this result we also find $\Psi(u) \to \Psi(in(d,u))$. Moreover it is easy to derive $\phi(v_0 \setminus s_0, g(s_0)) \to \phi(v_0, w_0)$, 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 $\phi(v_0, w_0)$. $\Box$

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

Proposition 8 (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 scheme 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 Eq3 and Test2, and subsequently the i.h.
$\leftarrow$: This time we use the second scheme 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:
If $test(d,t) = \mathsf{t}$ then it follows from $eq(in(d,u),t) = \mathsf{t}$ after application of axiom Eq3 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 3.1 we also have $in(d,u) = in(d,t)$. Since $test(d,t) = \mathsf{t}$ we can apply Proposition 3.4, and we conclude that $in(d,t) = t$. Combining these results yields

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

If $test(d,t) = \mathsf{f}$ then according to axiom Eq3 $eq(in(d,u),t) = \mathsf{f}$. 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 $\phi(s,t)$. □

It is also not hard to prove property formulas that contain the relational operators $\subset$ or $\subseteq$. As an example we give the following two identities.

Proposition 9 (Relational Set Identities). *For all $s, t : Set$ we have:*

1. $s \subset s = \mathsf{f}$,
2. $s \cap t \subseteq t = \mathsf{t}$.

Proof. 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. We use the first scheme for special induction. If $s \equiv s_0$ then axiom Sub1 can be applied. If $s \equiv in(d,t)$ then with the axioms Sub3 and Test2 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. We use the second scheme for special induction.

If $s \equiv s_0$ then $s \cap t \subseteq t = s_0 \cap t \subseteq t \overset{Sets7}{=} s_0 \subseteq t \overset{i}{=} \mathsf{t}$.

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

$$\begin{aligned} in(d,u) \cap t \subseteq t &\overset{Sets8}{=} in(d, u \cap t) \subseteq t \\ &\overset{3.4}{=} in(d, u \cap t) \subseteq in(d,t) \\ &\overset{ii}{=} rem(d, u \cap t) \subseteq rem(d,t) \\ &\overset{3.9}{=} rem(d,u) \cap rem(d,t) \subseteq rem(d,t). \end{aligned}$$

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

$$\begin{aligned} in(d,u) \cap t \subseteq t &\overset{3.1}{=} in(d, rem(d,u)) \cap t \subseteq t \\ &\overset{Sets8}{=} rem(d,u) \cap t \subseteq t \\ &\overset{3.5}{=} rem(d,u) \cap rem(d,t) \subseteq rem(d,t). \end{aligned}$$

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

We finish our specification of sets with two simple functions. The *size* function is needed for a 'neat' proof of the correctness of the third special induction scheme; it represents the number of elements in a set. The predicate *empty* is defined for cosmetic reasons. It is used in Section 4.

func	$size : Set \to Nat$	
	$empty : Set \to \mathbf{Bool}$	
rew	$size(s_0) = 0$	Size1
	$size(in(d,s)) = size(rem(d,s)) + 1$	Size2
	$empty(s) = eq(size(s), 0)$	Size3

Definition 10 (Special Set Induction 3). A proof of a property formula $\phi(s)$, by special set induction 3 on $s : Set$, is the deduction

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

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

Proposition 11. *Proofs by special set induction with the above scheme are correct.*

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

$$\begin{array}{ll} (i) & \phi(s_0), \\ (ii) & test(d,t) = \mathsf{t} \land \phi(rem(d,t)) \to \phi(t). \end{array}$$

The following results are needed:

$$\begin{array}{ll} (iii) & size(t) = 0 \to t = s_0, \\ (iv) & test(d,t) = \mathsf{t} \land size(t) = Sn \to size(rem(d,t)) = n, \end{array}$$

where $n : Nat$. Property formula (iii) is easily proved by standard induction on t, and (iv) follows with the help of Proposition 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 $\Phi(s,m) \equiv (size(s) = m \to \phi(s))$, 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 $\Phi(s,n) \to (test(d,t) = \mathsf{t} \wedge size(t) = Sn \to \phi(t))$. It follows by standard set induction on t that $\Phi(s,n) \to (size(t) = Sn \to \phi(t))$, and therefore that $\Phi(s,n) \to \Phi(s,Sn)$, and hence by induction we may conclude $\Phi(s,m)$. Finally we can substitute $size(s)$ for m, so that we find $\phi(s)$. □

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. We incorporate this idea in the following induction scheme.

Definition 12 (Special Set Induction 4). Let $f : Set \to D$ be a function with the property $eq(s,s_0) = \mathsf{f} \to test(f(s),s) = \mathsf{t}$. A proof of a property formula $\phi(s)$, by special set induction 4 on $s : Set$, is the deduction

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

where $t : Set$.

Proposition 13. *Proofs by special set induction with the above scheme are correct.*

Proof. Follows from the previous induction scheme. □

3 A rule for hybrid set induction

Thus far we gave induction schemes that concerned variables of a single sort. The careful reader may have noticed that correctness proofs of these schemes could be provided within the formal framework of μCRL, extended with some second order arguments, i.e. we had to reason about property formulas, using variables ϕ, ψ ranging over some universe of property formulas. Moreover, the proof we gave for Proposition 7 is not finite.

One can imagine that in many cases our schemes do not suffice for providing proofs on a more or less formal basis. As an example we have Proposition 21.1. In general, induction schemes where more than one variable varies in the induction step, see e.g. Definition 6, will be hard to prove on a formal basis. If we allow some 'meta-theoretical' reasoning, however, we can easily obtain a considerable generalisation of the standard scheme in the form of a rule.

We refer to the below rule as "hybrid" because it allows simultaneous reasoning on the structure of data terms and process terms. Here it is maybe interesting to note that μCRL has no rules for structural induction on process terms, although many useful results in process algebra have to be derived this way. We also remark that the need for hybrid induction once again illustrates the close interaction between data and processes.

Definition 14 (Hybrid Set Induction). Let $x, y_1, ..., y_n$ be data variables typed by $x, y_1 : S_1, y_2 : S_2, ..., y_n : S_n$ and let $f : S_1 \times ... \times S_n \rightarrow S_1$ be a function, or let $x, y_1, ...y_n$ be process variables and f a process operator. A proof of a property formula $\phi(s, x)$, by hybrid set induction on $s : Set$, is the deduction

$$\frac{\phi(s_0, y_1) \quad \phi(rem(d,t), f(y_1, ..., y_n)) \rightarrow \phi(in(d,t), y_1)}{\phi(s, x)}$$

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

Proposition 15. *Proofs by hybrid set induction are correct.*

Proof. We prove $\phi(s, x)$ from the above clauses. If $s \equiv s_0$ then this follows immediately. If $s \equiv in(e, u)$, where $e : D$ and $u : Set$, then if we prove $\phi(rem(e, u), f(y_1, ..., y_n))$ we are ready. Clearly, after another $size(rem(e, u))$ applications of the rem function we can obtain the empty set. Therefore, it suffices to prove $\phi(s_0, f^{size(s)}(y_1, ..., y_n))$ if s is not empty (we use $f^{n+2}(y_1, ..., y_n)$ to denote $f(f^{n+1}(y_1, ..., y_n), y'_2, ..., y'_n))$. This follows immediately from $\phi(s_0, y_1)$. $\square$

Note that if we adopt the rule for hybrid set induction, the first two schemes for special set induction in the previous section become superfluous.

4 Inductive proofs with sets in process algebra

In order to demonstrate the use of our induction schemes for sets in a different context, we study two special processes that can be regarded generalisations of the alternative composition, and parallel composition from process algebra. Moreover we demonstrate how our generalised parallel composition can be used for modelling a broadcast process.

4.1 Process algebra

We first recall the system PA_δ (Process Algebra with δ) from [BK84]. The signature of PA_δ has a set A of *atomic actions*, with typical elements $a, b, ...$, as a parameter. The special constant δ, or *deadlock*, represents the process that 'blocks'. We also have the binary operator $+$ for alternative composition, and $\cdot$ for sequential composition available. Concurrency is described by the operators $\|$ for *merge* or parallel composition, and the auxiliary operator $⌊\!⌊$ for *left-merge*. In term formation, brackets and variables from a set $x, y, z, ...$ are used. Letters $p, q, r, ...$ denote process terms, and brackets are omitted according to the convention that $\cdot$ binds stronger than any other operator, and that $+$ binds weaker than any other operator.

The axioms in Table 1 represent the axiom system PA_δ. The operator $+$ is commutative, associative and idempotent (A1 – A3). The operator $\cdot$ right distributes over $+$ and is associative (A4, A5). Furthermore δ behaves as the neutral element for $+$ (A6), and absorbs terms at the right side of $\cdot$ (A7). The operator $\|$ is defined in terms of the operator $⌊\!⌊$ (M1), and the axioms (M2 – M3) define the interaction between $⌊\!⌊$ and a, $a \cdot x$, and $x + y$, respectively.

(A1)	$x + y = y + x$	(M1)	$x \parallel y = x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x$
(A2)	$x + (y + z) = (x + y) + z$	(M2)	$a \mathbin{\|\!\_} x = a \cdot x$
(A3)	$x + x = x$	(M3)	$a \cdot x \mathbin{\|\!\_} y = a \cdot (x \parallel y)$
(A4)	$(x + y) \cdot z = x \cdot z + y \cdot z$	(M4)	$(x + y) \mathbin{\|\!\_} z = x \mathbin{\|\!\_} z + y \mathbin{\|\!\_} z$
(A5)	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$		
(A6)	$x + \delta = x$		
(A7)	$\delta \cdot x = \delta$		

Table 1: The axioms of PA_δ, where $a \in A \cup \{\delta\}$.

(Cond1)	$x \lhd \mathsf{t} \rhd y = x$
(Cond2)	$x \lhd \mathsf{f} \rhd y = y$
(Sum1)	$\Sigma_{d:D}(q) = q$
(Sum4)	$\Sigma_{d:D}(p + q) = \Sigma_{d:D}(p) + \Sigma_{d:D}(q)$
(Sum5)	$\Sigma_{d:D}(p \cdot q) = \Sigma_{d:D}(p) \cdot q$
(Sum6)	$\Sigma_{d:D}(p \mathbin{\|\!\_} q) = \Sigma_{d:D}(p) \mathbin{\|\!\_} q$
	provided d is not free in q in Sum1, Sum5 and Sum6

Table 2: Axioms for the conditional construct and Σ.

Associativity and commutativity of $+$ is denoted with AC, and associativity and commutativity of $\parallel$ is denoted with SC (Standard Concurrency, which is derivable for regular process expressions).

From μCRL we use the conditional construct and the Σ operator, which denotes a generalisation of $+$, e.g. the expression $\Sigma_{d:D}(P(d))$ denotes the (possibly infinite) sum of processes P, parameterised with elements from the sort D. The necessary axioms are given in Table 2. The binding strength of $.\lhd b \rhd .$ is taken between $+$ and the other operators of PA_δ. If we prove identities with conditionals using case distinction, we usually annotate this with CD.

4.2 Generalised sums

We define a generalised sum process S_P, not to be confused with the Σ operator. Then we prove some properties of this process, and demonstrate how it can be defined in terms of Σ.

Definition 16 (Generalised Sums). Let $P(d : D)$ be a given process, and $f : \mathit{Set} \to D$ a function with the property $eq(s, s_0) = \mathsf{f} \to \mathit{test}(f(s), s) = \mathsf{t}$. A generalised sum for the process $P(d : D)$ over a set s is

$$S_P(s : \mathit{Set}) \stackrel{\text{def}}{=} \delta \lhd \mathit{empty}(s) \rhd (P(f(s)) + S_P(\mathit{rem}(f(s), s))).$$

For example, if D represents the natural numbers and $f(s)$ the minimum in s then it is easy to derive $S_P(in(2, in(5, in(1, s_0)))) = P(1) + (P(2) + P(5))$.

Proposition 17 (Sets and Sums). *Let $d : D$ and $s, t : Set$. For generalised sums for a process $P(d : D)$ we have:*

1. $S_P(s_0) = \delta$,
2. $S_P(in(d, s)) = P(d) + S_P(rem(d, s))$,
3. $S_P(s \cup t) = S_P(s) + S_P(t)$.

Proof.

1. Immediately from the definition of S_P.

2. By induction on s. We use the scheme in Definition 12.

   If $s \equiv s_0$ then $S_P(in(d, s)) = P(d) + S_P(s) \overset{Sets3}{=} P(d) + S_P(rem(d, s))$.

   Our i.h. is $S_P(in(d, rem(f(s), s))) = P(d) + S_P(rem(d, rem(f(s), s)))$, and we use the following intermediate results: If $eq(f(in(d, s)), d) = \mathsf{f}$ then

$$\begin{array}{ll} (i) & f(in(d, s)) = f(s), \\ (ii) & f(in(d, s)) = f(rem(d, s)), \end{array}$$

   which follows easily from the properties of f and the *test* function.

   If $eq(f(in(d, s)), d) = \mathsf{t}$ then Proposition 17.2 follows directly from the definition of S_P.

   If $eq(f(in(d, s)), d) = \mathsf{f}$ then we derive

$$\begin{array}{rcl} S_P(in(d, s)) & = & P(f(in(d, s))) + S_P(rem(f(in(d, s)), in(d, s))) \\ & \overset{Sets4}{=} & P(f(in(d, s))) + S_P(in(d, rem(f(in(d, s)), s))) \\ & \overset{i,ii}{=} & P(f(rem(d, s))) + S_P(in(d, rem(f(s), s))) \\ & \overset{i.h.}{=} & P(f(rem(d, s))) + (P(d) + S_P(rem(d, rem(f(s), s)))) \\ & \overset{3.3}{=} & P(f(rem(d, s))) + (P(d) + S_P(rem(f(s), rem(d, s)))) \\ & \overset{i,ii,AC}{=} & P(d) + (P(f(rem(d, s))) + S_P(rem(f(rem(d, s)), rem(d, s)))) \\ & = & P(d) + S_P(rem(d, s)). \end{array}$$

3. Also by induction on s. This time we use the scheme in Definition 6.

   If $s \equiv s_0$ then $S_P(s \cup t) \overset{Sets5}{=} S_P(t) \overset{A6,AC}{=} \delta + S_P(t) \overset{17.1}{=} S_P(s) + S_P(t)$.

   Our i.h. is $S_P(rem(d, s) \cup rem(d, t)) = S_P(rem(d, s)) + S_P(rem(d, t))$.

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

$$
\begin{array}{rcl}
S_P(in(d,s) \cup t) & \stackrel{Sets6}{=} & S_P(in(d, s \cup t)) \\
& \stackrel{17.2}{=} & P(d) + S_P(rem(d, s \cup t)) \\
& \stackrel{3.8}{=} & P(d) + S_P(rem(d,s) \cup rem(d,t)) \\
& \stackrel{i.h.}{=} & P(d) + (S_P(rem(d,s)) + S_P(rem(d,t))) \\
& \stackrel{A3,AC}{=} & (P(d) + S_P(rem(d,s))) + (P(d) + S_P(rem(d,t))) \\
& \stackrel{17.2}{=} & S_P(in(d,s)) + S_P(in(d,t)) \\
& \stackrel{3.4}{=} & S_P(in(d,s)) + S_P(t).
\end{array}
$$

If $test(d,t) = \mathsf{f}$ we simply follow a different route after application of the i.h.:

$$
\begin{array}{rcl}
S_P(in(d,s) \cup t) & = & \ldots \stackrel{i.h}{=} P(d) + (S_P(rem(d,s)) + S_P(rem(d,t))) \\
& \stackrel{AC}{=} & (P(d) + S_P(rem(d,s))) + S_P(rem(d,t)) \\
& \stackrel{17.2}{=} & S_P(in(d,s)) + S_P(rem(d,t)) \\
& \stackrel{3.5}{=} & S_P(in(d,s)) + S_P(t).
\end{array}
$$

□

The first two identities of the above proposition imply that we can argue about S_P without using f. By definition of constructor sets we know that any process term $S_P(t)$ can be rewritten by either the first or the second identity. The third identity shows how set composition interacts with the generalised sum process. This correspondence is due to the fact that both $+$ and $\cup$ are idempotent, e.g.

$$S_P(s) = S_P(s \cup s) \stackrel{17.3}{=} S_P(s) + S_P(s) \stackrel{A3}{=} S_P(s).$$

The above identities can be also be proven correct using properties of Σ, which would allow here for easier proofs. (This is not the case, however, with the generalised merge.) Using the axioms for Σ, it can be proved that $S_P(s)$ is a summand of $\Sigma_{d:D}(P(d))$. If s contains all elements from D then they are even equal.

Proposition 18 (Sums and Σ). *Let $s : Set$. For generalised sums for a process $P(d : D)$ we have:*

$$S_P(s) = \Sigma_{d:D}(P(d) \triangleleft test(d,s) \triangleright \delta).$$

Proof. We need some basic identities for Σ:

$$
\begin{array}{ll}
(i) & \Sigma_{d:D}(p \triangleleft test(d, s_0) \triangleright \delta) = \delta, \\
(ii) & \Sigma_{d:D}(p \triangleleft eq(d,e) \triangleright \delta) = p[e/d], \\
(iii) & \Sigma_{d:D}(p \triangleleft test(d, in(e,s)) \triangleright \delta) = \Sigma_{d:D}(p \triangleleft test(d, rem(e,s)) \triangleright \delta) + p[e/d],
\end{array}
$$

where $p[e/d]$ denotes the process term p with $e : D$ substituted for d. Identity (i) follows using axiom Test1 and axiom Sum1, (ii) is proved in [GK92], and (iii) is proved as follows:

$$
\begin{array}{rl}
\multicolumn{2}{l}{\Sigma_{d:D}(p \lhd test(d, in(e,s)) \rhd \delta)} \\
\stackrel{3.1,Test2}{=} & \Sigma_{d:D}(p \lhd or(eq(d,e), test(d, rem(e,s))) \rhd \delta) \\
\stackrel{CD}{=} & \Sigma_{d:D}(p \lhd test(d, rem(e,s)) \rhd \delta + p \lhd eq(e,s) \rhd \delta) \\
\stackrel{Sum4,ii}{=} & \Sigma_{d:D}(p \lhd test(d, rem(e,s)) \rhd \delta) + p[e/d].
\end{array}
$$

Proposition 18 now follows by induction on s, using the first scheme for special set induction. If $s \equiv s_0$ the result follows trivially from (i), and if $s \equiv in(e,t)$ then use (iii) and apply the i.h. □

4.3 Generalised merges

The generalised merge M_P as defined below is inspired by [Mil89], where it was used for the specification and verification of 'Milner's scheduler'; a simple communication network with a large amount of states, and therefore often used for testing the performance of tools.

Definition 19 (Generalised Merges). Let $P(d:D)$ be a given process, and $f: Set \to D$ a function with the property $eq(s, s_0) = \mathsf{f} \to test(f(s), s) = \mathsf{t}$. A generalised merge for the process $P(d:D)$ over a set s is

$$
\begin{array}{rl}
M_P(s: Set) \stackrel{\text{def}}{=} & \delta \lhd empty(s) \rhd \\
& (P(f(s)) \lhd empty(rem(f(s), s)) \rhd P(f(s)) \parallel M_P(rem(f(s), s))).
\end{array}
$$

For example, if D represents the natural numbers and $f(s)$ the minimum in s then it is easy to derive $M_P(in(5, in(2, in(5, in(1, s_0))))) = P(1) \parallel (P(2) \parallel P(5))$.

Proposition 20 (Sets and Merges). *Let $d:D$ and $s,t: Set$. For generalised merges for a process $P(d:D)$ we have:*

1. $M_P(s_0) = \delta$,
2. $M_P(in(d,s)) = P(d) \lhd empty(rem(d,s)) \rhd P(d) \parallel M_P(rem(d,s))$,
3. $empty(s) = \mathsf{f} \wedge empty(t) = \mathsf{f} \wedge empty(s \cap t) = \mathsf{t} \to M_P(s \cup t) = M_P(s) \parallel M_P(t)$.

Proof.

1. Immediately from the definition of M_P.
2. Analogously to the proof of Proposition 17.2. In this proof the axioms for SC are needed instead of AC.
3. By induction on s. We use the scheme in Definition 4. Moreover we use the following intermediate results: If $empty(t) = \mathsf{f} \wedge empty(in(d,s) \cap t) = \mathsf{t}$ then

$$
\begin{array}{ll}
(i) & empty(rem(d, s \cup t)) = \mathsf{f}, \\
(ii) & empty(rem(d,s) \cap t) = \mathsf{t}, \\
(iii) & rem(d,t) = t.
\end{array}
$$

If $s \equiv s_0$ the result follows trivially. Our i.h. is $empty(rem(d,s)) = \mathsf{f} \wedge empty(t) = \mathsf{f} \wedge empty(rem(d,s) \cap t) = \mathsf{t} \rightarrow M_P(rem(d,s) \cup t) = M_P(rem(d,s)) \parallel M_P(t)$. We assume that $empty(t) = \mathsf{f} \wedge empty(in(d,s) \cap t) = \mathsf{t}$, so

$$\begin{aligned} M_P(in(d,s) \cup t) &\stackrel{Sets6}{=} M_P(in(d, s \cup t)) \\ &\stackrel{20.2,i}{=} P(d) \parallel M_P(rem(d, s \cup t)) \\ &\stackrel{3.8,iii}{=} P(d) \parallel M_P(rem(d,s) \cup t). \end{aligned}$$

If $empty(rem(d,s)) = \mathsf{t}$ then $rem(d,s) = s_0$, and according to Proposition 20.2 $P(d) = M_P(in(d,s))$, so in this case we are ready.

If $empty(rem(d,s)) = \mathsf{f}$ then we can combine (ii), together with our assumption $empty(t) = \mathsf{f}$, with the i.h.:

$$\begin{aligned} P(d) \parallel M_P(rem(d,s) \cup t) &\stackrel{i.h.}{=} P(d) \parallel (M_P(rem(d,s)) \parallel M_P(t)) \\ &\stackrel{SC}{=} (P(d) \parallel M_P(rem(d,s))) \parallel M_P(t) \\ &\stackrel{20.2}{=} M_P(in(d,s)) \parallel M_P(t). \end{aligned}$$

□

The first two identities of the above proposition imply that we can argue about any process term M_P without using f. The third property formula implies that if $t_1 \cup t_2 \cup ... \cup t_n$ is a partition of a set s then

$$M_P(s) = M_P(t_1) \parallel (M_P(t_2) \parallel (\; ... \parallel M_P(t_n) \; ... \;)).$$

As we argued above, the relation between S_P and Σ is rather straightforward. We now show that this is also the case with M_P and Σ; there are some very interesting relations, and again we can use a variety of induction principles for the proofs.

Proposition 21 (Merges and Σ). *Let $s : Set$. For generalised merges for a process $P(d : D)$ and an atomic action $a(d : D)$ we have:*

1. $size(s) > 0 = \mathsf{t} \rightarrow x \parallel M_P(s) = \Sigma_{d:D}(x \parallel M_P(s) \lhd test(d,s) \rhd \delta)$,
2. $M_P(s) = \Sigma_{d:D}(M_P(s) \lhd test(d,s) \rhd \delta)$,
3. $M_a(s) = \Sigma_{d:D}((a(d) \lhd empty(rem(d,s)) \rhd a(d) \cdot M_a(rem(d,s))) \lhd test(d,s) \rhd \delta)$.

Proof.

1. For a proof of this property formula we use the rule for hybrid set induction, defined in Section 3. Moreover we use the following identity:

$$(i) \quad (p \lhd empty(s) \rhd q) \lhd test(d,s) \rhd \delta = q \lhd test(d,s) \rhd \delta.$$

We apply induction on s, and $f(x,y) \stackrel{\text{def}}{=} x \parallel y$. So if $s \equiv in(e,t)$, where $e : D$ and $t : Set$, and if $\phi(s,x)$ is defined as property formula 21.1, the i.h. is $\phi(rem(e,t), x \parallel y)$.

The clause $\phi(s_0, x)$ follows trivially. Furthermore we derive:

$$\begin{aligned}
&\Sigma_{d:D}(x \parallel M_P(in(e,t)) \lhd test(d, in(e,t)) \rhd \delta) \\
&\stackrel{18.iii}{=} \quad \Sigma_{d:D}(x \parallel M_P(in(e,t)) \lhd test(d, rem(e,t)) \rhd \delta) \ + x \parallel M_P(in(e,t)) \\
&\stackrel{20.2,CD}{=} \quad \Sigma_{d:D}((x \parallel P(e) \lhd empty(rem(e,t)) \rhd x \parallel (P(e) \parallel M_P(rem(e,t)))) \\
&\qquad \lhd test(d, rem(e,t)) \rhd \delta) \ + x \parallel M_P(in(e,t)) \\
&\stackrel{i,SC}{=} \quad \Sigma_{d:D}((x \parallel P(e)) \parallel M_P(rem(e,t)) \lhd test(d, rem(e,t)) \rhd \delta) \ + \\
&\qquad x \parallel M_P(in(e,t)).
\end{aligned}$$

If $empty(rem(e,t)) = \mathsf{t}$ we can apply 18.i, and if $empty(rem(e,t)) = \mathsf{f}$ we can apply the i.h.

2. By standard set induction on s. The previous identity can be used.

3. We use the first scheme for special set induction. If $s \equiv s_0$ the result follows using Proposition 18.i. We take $s \equiv in(e,t)$:

$$\begin{aligned}
M_a(in(e,t)) &\stackrel{20.2}{=} a(e) \lhd empty(rem(e,t)) \rhd a(e) \parallel M_a(rem(e,t)) \\
&\stackrel{M1,M2}{=} a(e) \lhd empty(rem(e,t)) \rhd (a(e) \cdot M_a(rem(e,t)) \ + \\
&\qquad M_a(rem(e,t)) \mathbin{\lfloor\!\lfloor} a(e)) \\
&\stackrel{i.h.,Sum6}{=} a(e) \lhd empty(rem(e,t)) \rhd (a(e) \cdot M_a(rem(e,t)) \ + \\
&\qquad \Sigma_{d:D}(((a(d) \lhd empty(rem(d, rem(e,t))) \rhd \\
&\qquad\quad a(d) \cdot M_a(rem(d, rem(e,t)))) \lhd test(d, rem(e,t)) \rhd \delta) \mathbin{\lfloor\!\lfloor} a(e))) \\
&\stackrel{CD,M2,M3}{=} a(e) \lhd empty(rem(e,t)) \rhd (a(e) \cdot M_a(rem(e,t)) \ + \\
&\qquad \Sigma_{d:D}((a(d) \cdot a(e) \lhd empty(rem(d, rem(e,t))) \rhd \\
&\qquad\quad a(d) \cdot (a(e) \parallel M_a(rem(d, rem(e,t))))) \lhd test(d, rem(e,t)) \rhd \delta)) \\
&\stackrel{CD,3.3,20.2}{=} a(e) \lhd empty(rem(e,t)) \rhd (a(e) \cdot M_a(rem(e,t)) \ + \\
&\qquad \Sigma_{d:D}(a(d) \cdot M_a(in(e, rem(d,t))) \lhd test(d, rem(e,t)) \rhd \delta)) \\
&\stackrel{CD,3.10}{=} a(e) \lhd empty(rem(e,t)) \rhd a(e) \cdot M_a(rem(e,t)) \ + \\
&\qquad \Sigma_{d:D}(a(d) \cdot M_a(rem(d, in(e,t))) \lhd test(d, rem(e,t)) \rhd \delta) \\
&\stackrel{CD,Sets4,3.10}{=} a(e) \lhd empty(rem(e, in(e,t))) \rhd a(e) \cdot M_a(rem(e, in(e,t))) \ + \\
&\qquad \Sigma_{d:D}((a(d) \lhd empty(rem(d, in(e,t))) \rhd a(d) \cdot M_a(rem(d, in(e,t)))) \\
&\qquad\quad \lhd test(d, rem(e,t)) \rhd \delta) \\
&\stackrel{18.iii}{=} \Sigma_{d:D}((a(d) \lhd empty(rem(d, in(e,t))) \rhd a(d) \cdot M_a(rem(d, in(e,t)))) \\
&\qquad \lhd test(d, in(e,t)) \rhd \delta).
\end{aligned}$$

□

4.4 Broadcasting

The process in Definition 22 models a *broadcasting* mechanism in μCRL. It is inspired by [Bru93], and it can be interpreted as follows: there is a set D that contains all possible sources, or senders of messages, in a communication network. As soon as the network is activated, the broadcast process waits for a message. If a message m comes in from some source d, it is sent to all the destinations in some fixed set E, except d. After this is finished, the process waits again for a message.

Definition 22 (Broadcast Process). Let $send(d : D, m : Msg), read(d : D, m : Msg)$ be atomic actions, where Msg is some non-empty message set, and $E : Set$ a fixed set with elements from D. A broadcast process for a set s of destinations, and a source m is

$$\begin{array}{lcl} READ & \stackrel{\text{def}}{=} & \Sigma_{d:D,m:Msg}(read(d,m) \cdot BC(rem(d,E),m)) \\ BC(s : Set, m : Msg) & \stackrel{\text{def}}{=} & \Sigma_{d:D}(send(d,m) \cdot BC(rem(d,s),m) \lhd test(d,s) \rhd \delta) + \\ & & READ \lhd empty(s) \rhd \delta. \end{array}$$

The actual broadcasting must be regarded as the successive sending of messages to all destinations in the set s. The following equation captures the intuition that broadcasting can be modelled as a parallel process.

Proposition 23 (Broadcasting and Merges). *Let M_{send_m} denote the generalised merge for the atomic action $send(d : D, m : Msg)$, then for a broadcast process for a set s of destinations, and a source m we have:*

$$BC(s,m) = READ \lhd empty(s) \rhd M_{send_m}(s) \cdot READ.$$

Proof. By induction on s, according to the induction scheme in Definition 10. For $s \equiv s_0$ the result follows easily. Our i.h. is

$$\begin{array}{l} test(d,s) = \mathsf{t} \wedge \\ BC(rem(d,s),m) = READ \lhd empty(rem(d,s)) \rhd M_{send_m}(rem(d,s)) \cdot READ, \end{array}$$

and we derive

$$\begin{array}{lcl} BC(s,m) & \stackrel{i.h.,CD}{=} & \Sigma_{d:D}(send(d,m) \cdot \\ & & \quad (READ \lhd empty(rem(d,s)) \rhd M_{send_m}(rem(d,s)) \cdot READ) \\ & & \qquad \lhd test(d,s) \rhd \delta) + READ \lhd empty(s) \rhd \delta \\ & \stackrel{CD}{=} & \Sigma_{d:D}((send(d,m) \lhd empty(rem(d,s)) \rhd \\ & & \qquad send(d,m) \cdot M_{send_m}(rem(d,s))) \cdot READ \\ & & \qquad \lhd test(d,s) \rhd \delta) + READ \lhd empty(s) \rhd \delta \\ & \stackrel{CD,Sum5}{=} & \Sigma_{d:D}((send(d,m) \lhd empty(rem(d,s)) \rhd \\ & & \qquad send(d,m) \cdot M_{send_m}(rem(d,s))) \\ & & \qquad \lhd test(d,s) \rhd \delta) \cdot READ + READ \lhd empty(s) \rhd \delta \\ & \stackrel{21.3}{=} & M_{send_m}(s) \cdot READ + READ \lhd empty(s) \rhd \delta. \end{array}$$

The result now follows easily by case distinction on $empty(s)$. □

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.

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

[Bru93] J.J. Brunekreef, J.P. Katoen, R.L.C. Koymans and S. Mauw. Design and analysis of dynamic leader election protocols in broadcast networks. Technical report P9324, University of Amsterdam, 1993.

[GK92] J.F. Groote and H. Korver. A correctness proof of the bakery protocol in μCRL. Technical Report 80, Logic Group Preprint Series, Utrecht University, 1992.

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

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

[GP94] J.F. Groote and A. Ponse. Proof theory for μCRL: 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.

[GvW94] J.F. Groote and J.J. van Wamel. Algebraic data types and induction in μCRL. Report P9409, University of Amsterdam, 1994.

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

[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.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

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

[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.