Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

A note on n similar parallel processes

J.F. Groote

Computer Science/Department of Software Technology

CS-R9626 1996

Report CS-R9626
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

A Note on n Similar Parallel Processes

Jan Friso Groote
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
E-mail: jfg@cwi.nl

Abstract

We show that defining a finite but unbounded number of parallel processes using the equation $S(k,dt) = P(0,get(0,dt)) \lhd eq(k,0) \rhd (S(k-1,dt) \parallel P(k,get(k,dt)))$ is well defined, if one adopts the principle CL-RSP. We also provide means to easily derive a linear process equation with the same behaviour as $S(k,dt)$.

CR Subject Classification (1991): F.2: Analysis of Algorithms and Problem; F.4: Mathematical Logic and Formal Languages.
AMS Subject Classification (1991): 68Q60: Specification and verification of programs; 68Q22: Parallel and distributed algorithms.
Keywords & Phrases: Parallelism, Expansion Theorem, μCRL.

1 Introduction

Distributed algorithms are generally configured as an arbitrarily large but finite set of processors that run a similar program. Using the formalism μCRL [8] this can be neatly described. Assume that the individual processes are given by $P(k)$, where $k \in \mathbb{N}$ is the index of the process. The following equation puts n of these processes in parallel:

$$S(n{:}\mathbb{N}) = P(0) \lhd eq(n,0) \rhd (S(n-1) \parallel P(n)). \tag{1}$$

Clearly, the process $S(n)$ stands for $P(0) \parallel P(1) \parallel \ldots \parallel P(n)$.

We find descriptions along this line in verifications of the bakery protocol [6], Milner's scheduler [12], a leader election protocol [5], grid protocols [2] and a summing protocol [9].

The description in equation (1) gives rise to two issues. The first one is whether the equation *defines* in a proper way that S is the parallel composition of the processes $P(k)$. It is clear that the parallel composition of processes $P(k)$ is a solution for S. In this note we show that, assuming the principle CL-RSP (Convergent Linear Recursive Specification Principle, [3]), this is the only solution for S in (1). So, an equation in the form of (1) is indeed a proper definition.

The second issue is to transform the description in (1) to a format that is more convenient for verification purposes. Our experience currently indicates that the so called *linear format* is an adequate basis for verifications. We show in general terms how given a linear format for the processes $P(k)$, a process in linear format equivalent to $S(n)$ can be given.

The 'expansion' Theorem 3.5 that we provide is actually rather straightforward. However, it puts stress on several details that are easily overlooked when trying to carry out the tedious act of linearisation of a set of processes without the help of such general theorems. This is confirmed by the proof of Theorem 3.5, as it requires the formulation of Lemma 3.2 which can be proven by induction.

We think that readers will find Theorem 3.5 a convenient tool for the verification of distributed systems with an unbounded but finite number of processes. We give an extensive exposition of a simple example as an illustration of the use of Theorem 3.5.

We assume that the reader has a basic knowledge of process algebra, μCRL, its proof theory and some of its proof techniques (see e.g. [1, 7, 8, 11]).

2 Preliminaries

We assume the existence of a sort **Bool** of booleans with constants true t and false f, and the standard operators $\neg$, $\vee$ $\wedge$ and $\rightarrow$. When required we explicitly use the fact that there are exactly two booleans.

We also assume the existence of the sort of natural numbers $\mathbb{N}$ with the standard constants 0, 1, 2, etc. and standard operations such as $+$, $-$, eq (equality), $\leq$, $\geq$, $>$ and $<$. We use standard induction on natural numbers.

We rely on the notion of *a linear process*. It is useful to bring the notion of a linear process to the level of a *Linear Process Operator* (*LPO*), as this allows a more convenient formulation of the *Recursive Specification Principle*.

Definition 2.1. A *Linear Process Operator* (*LPO*) Ψ is an expression of the form:

$$\lambda p{:}D \rightarrow \mathbb{P}.\lambda d{:}D.\sum_{i \in I}\sum_{e_i:D_i} a_i(f_i(d,e_i))\, p(g_i(d,e_i)) \lhd c_i(d,e_i) \rhd \delta$$

for some finite index set I, actions a_i, data types D_i and D_{a_i}, functions $f_i : D \rightarrow D_i \rightarrow D_{a_i}$, $g_i : D \rightarrow D_i \rightarrow D$ and $c_i : D \rightarrow D_i \rightarrow \mathbf{Bool}$. $\mathbb{P}$ is the sort of processes.

Note that in general, processes have more than one parameter. Using pairing and projection functions, this is easily seen to be an non essential extension. In [3] linear processes were also equipped with a termination option. For reasons of conciseness, we omit it here. A last remark is that from an LPO Ψ one can easily derive its associated Linear Process Equation $p(d) = \Psi pd$. The other way, from LPE to LPO is comparably easy. Henceforth, we use LPOs and LPEs interchangably.

Definition 2.2. A linear process operator Ψ written in the form above is called *convergent* iff there is a well-founded ordering $<$ on D such that $g_i(d,e_i) < d$ for all $d \in D$, $i \in I$ and $e_i \in D_i$ with $a_i = \tau$ and $c_i(d,e_i)$.

We assume the validity of the following principles, which are restricted variations from the corresponding principles in [3].

Definition 2.3. The *Recursive Definition Principle* (*RDP*) says that every linear process operator Ψ has at least one fixed point, i.e. there exists a $p : D \rightarrow \mathbb{P}$ such that $p = \Psi p$.

The *Convergent Linear Recursive Specification Principle* (*CL-RSP*) says that every convergent linear process operator has at most one fixed point, i.e. for all $p : D \to \mathbb{P}$ and $q : D \to \mathbb{P}$ if $p = \Psi p$ and $q = \Psi q$, then $p = q$.

3 Linearisation of parallel processes

3.1 Definition

We provide here the linearisation of n linear processes $P(k, d)$. The natural number k ($0 \leq k \leq n$) is the index of the process and the parameter d of some arbitrary sort D denotes other parameters. We assume that each process $P(k, d)$ is defined using the following linear equation:

$$P(k{:}\mathbb{N}, d{:}D) = \sum_{i \in I} \sum_{e_i:E_i} a_i(f_i(k, d, e_i))\, P(k, g_i(k, d, e_i)) \lhd c_i(k, d, e_i) \rhd \delta. \tag{2}$$

We also assume that this equation is convergent, as this guarantees that this equation defines a unique process.

In order to define the parallel composition we need a new sort $DTable$, which is a table indexed by natural numbers containing elements of the sort D. In order to do so, we also need an auxiliary function $if : \mathbf{Bool} \times D \times D \to D$ reflecting $if - then - else$. In the sequel we also use a function $eq : D \times D \to \mathbf{Bool}$, expressing equality of the elements in D. We do not explicitly provide defining equations for these functions.

The constant emT of sort $DTable$ is the empty table. The function upd enters a new entry in the table and the function get gets a specific entry from the table. We characterise these operators by one single equation. Note that we do not specify what happens if an element from the empty table is being read. We refer to the characterising axiom for tables as the *table axiom*. Besides this axiom, we use the fact that tables are constructed using the empty table and the update function. This allows us the use of induction on these two operations.

sort	$DTable$
func	$emT :\to DTable$
	$upd : \mathbb{N} \times D \times DTable \to DTable$
	$get : \mathbb{N} \times DTable \to D$
var	$n, m : \mathbb{N}$, $d, d'{:}D$, $dt{:}DTable$
rew	$get(n, upd(m, d, dt)) = if(eq(n, m), d, get(n, dt))$

We can use the following process definition to put n of the processes $P(k, d)$ in parallel.

$$S(n{:}\mathbb{N}, dt{:}DTable) = P(0, get(0, dt)) \lhd (n, 0) \rhd (P(n, get(n, dt)) \parallel S(n - 1, dt)). \tag{3}$$

We assume that there is a commutative and associative communication function γ that explains how two actions can synchronise. In case two actions do not synchronise it yields δ. In this note we assume the so-called handshaking axiom, that says that no more than two actions can synchronise. In other words, for all actions a_1, a_2 and a_3, $\gamma(a_1, \gamma(a_2, a_3)) = \delta$ (cf. [1]). We also assume that the internal action τ cannot communicate; i.e. $\gamma(\tau, a) = \delta$.

3.2 Expansion

In this section we work towards a linear description of $S(n, dt)$ (Lemma 3.2). As a bonus we get that $S(n, dt)$ has at most one solution (Corollary 3.4). We also provide an alternative transformed linear description which we believe to be more convenient to be used in concrete instances (Theorem 3.5).

The following lemma gives a number of auxiliary facts that are used in the calculations to follow.

Lemma 3.1.

1. $\neg(c_1 \wedge c_2) \rightarrow if(c_1, d_1, if(c_2, d_2, d_3)) = if(c_2, d_2, if(c_1, d_1, d_3))$;
2. $m > n \rightarrow S(n, dt) = S(n, upd(m, d, dt))$;
3. $m_1 \neq m_2 \rightarrow upd(m_1, d_1, upd(m_2, d_2, dt)) = upd(m_2, d_2, upd(m_1, d_1, dt))$;

Proof. These facts are straightforwardly proven by induction on c_1 and c_2, n and dt, respectively. □

Below we present the main Lemma of this note. It gives an expansion of S. As has been stated above, we find this form not very convenient as it has the condition $k_1 > k_2$ in its second group of summands. The more convenient form in Theorem 3.5 has $i_1 \geq i_2$ as an alternative. But contrary to the linearisation in Theorem 3.5 we can prove this Lemma with induction on n.

Lemma 3.2. *The process S as defined in (3) is a solution for Q in equation (4) below. The set I and the functions f_i, g_i and c_i are those that occur in equation (2).*

$$
\begin{aligned}
&Q(n{:}\mathbb{N}, dt{:}DTable) = \\
&\quad \textstyle\sum_{i\in I} \sum_{k:\mathbb{N}} \sum_{e_i:E_i} a_i(f_i(k, get(k, dt), e_i))\, Q(n, upd(k, g_i(k, get(k, dt), e_i), dt)) \\
&\qquad \triangleleft c_i(k, get(k, dt), e_i) \wedge k \leq n \triangleright \delta + \\
&\quad \textstyle\sum_{i_1\in I} \sum_{i_2\in I} \sum_{k_1:\mathbb{N}} \sum_{k_2:\mathbb{N}} \sum_{e_{i_1}:E_{i_1}} \sum_{e_{i_2}:E_{i_2}} \gamma(a_{i_1}, a_{i_2})(f_{i_1}(k_1, get(k_1, dt), e_{i_1})) \\
&\qquad Q(n, upd(k_1, g_{i_1}(k_1, get(k_1, dt), e_{i_1}), upd(k_2, g_{i_2}(k_2, get(k_2, dt), e_{i_2}), dt))) \\
&\qquad\quad \triangleleft c_{i_1}(k_1, get(k_1, dt), e_{i_1}) \wedge c_{i_2}(k_2, get(k_2, dt), e_{i_2}) \wedge \\
&\qquad\quad eq(f_{i_1}(k_1, get(k_1, dt), e_{i_1}), f_{i_2}(k_2, get(k_2, dt), e_{i_2})) \wedge k_1 > k_2 \wedge k_1 \leq n \triangleright \delta.
\end{aligned}
\tag{4}
$$

Proof. So, we must show the following equation to hold:

$$
\begin{aligned}
&S(n, dt) = \\
&\quad \textstyle\sum_{i\in I} \sum_{k:\mathbb{N}} \sum_{e_i:E_i} a_i(f_i(k, get(k, dt), e_i))\, S(n, upd(k, g_i(k, get(k, dt), e_i), dt)) \\
&\qquad \triangleleft c_i(k, get(k, dt), e_i) \wedge k \leq n \triangleright \delta + \\
&\quad \textstyle\sum_{i_1\in I} \sum_{i_2\in I} \sum_{k_1:\mathbb{N}} \sum_{k_2:\mathbb{N}} \sum_{e_{i_1}:E_{i_1}} \sum_{e_{i_2}:E_{i_2}} \gamma(a_{i_1}, a_{i_2})(f_{i_1}(k_1, get(k_1, dt), e_{i_1})) \\
&\qquad S(n, upd(k_1, g_{i_1}(k_1, get(k_1, dt), e_{i_1}), upd(k_2, g_{i_2}(k_2, get(k_2, dt), e_{i_2}), dt))) \\
&\qquad\quad \triangleleft c_{i_1}(k_1, get(k_1, dt), e_{i_1}) \wedge c_{i_2}(k_2, get(k_2, dt), e_{i_2}) \wedge \\
&\qquad\quad eq(f_{i_1}(k_1, get(k_1, dt), e_{i_1}), f_{i_2}(k_2, get(k_2, dt), e_{i_2})) \wedge k_1 > k_2 \wedge k_1 \leq n \triangleright \delta.
\end{aligned}
\tag{5}
$$

We do this with induction on n.
Base. In case $n = 0$, we can replace occurrences of $S(0, dt)$ by $P(0, get(0, dt))$ according to definition (3). Using the sum elimination lemma [6, 11], the first summand at the right hand side simplifies, as we can take $k = 0$. Furthermore, the second summand disappears altogether, because there does not exist a $k_2 < k_1 \leq 0$. So, the equation above simplifies to:

$$\begin{aligned}&P(0, get(0, dt)) = \\ &\quad \textstyle\sum_{i \in I} \sum_{e_i : E_i} a_i(f_i(0, get(0, dt), e_i))\, P(0, get(0, upd(0, g_i(0, get(0, dt), e_i), dt))) \\ &\qquad \lhd c_i(0, get(0, dt), e_i) \rhd \delta.\end{aligned}$$

This equation simplifies even further to

$$\begin{aligned}&P(0, get(0, dt)) = \\ &\quad \textstyle\sum_{i \in I} \sum_{e_i : E_i} a_i(f_i(0, get(0, dt), e_i))\, P(0, g_i(0, get(0, dt), e_i)) \lhd c_i(0, get(0, dt), e_i) \rhd \delta,\end{aligned}$$

which is an instantiation of the defining equation (2).
Induction step. Suppose equation (5) holds for some $n \geq 0$. We show that it also holds for $n + 1$. So, we must show that

$$\begin{aligned}&S(n+1, dt) = \\ &\quad \textstyle\sum_{i \in I} \sum_{k : \mathbb{N}} \sum_{e_i : E_i} a_i(f_i(k, get(k, dt), e_i))\, S(n+1, upd(k, g_i(k, get(k, dt), e_i), dt)) \\ &\qquad \lhd c_i(k, get(k, dt), e_i) \wedge k \leq n+1 \rhd \delta + \\ &\quad \textstyle\sum_{i_1 \in I} \sum_{i_2 \in I} \sum_{k_1 : \mathbb{N}} \sum_{k_2 : \mathbb{N}} \sum_{e_{i_1} : E_{i_1}} \sum_{e_{i_2} : E_{i_2}} \gamma(a_{i_1}, a_{i_2})(f_{i_1}(k_1, get(k_1, dt), e_{i_1})) \qquad (6) \\ &\qquad S(n+1, upd(k_1, g_{i_1}(k_1, get(k_1, dt), e_{i_1}), upd(k_2, g_{i_2}(k_2, get(k_2, dt), e_{i_2}), dt))) \\ &\qquad\quad \lhd c_{i_1}(k_1, get(k_1, dt), e_{i_1}) \wedge c_{i_2}(k_2, get(k_2, dt), e_{i_2}) \wedge \\ &\qquad\quad eq(f_{i_1}(k_1, get(k_1, dt), e_{i_1}), f_{i_2}(k_2, get(k_2, dt), e_{i_2})) \wedge k_1 > k_2 \wedge k_1 \leq n+1 \rhd \delta.\end{aligned}$$

By definition $S(n + 1, dt)$ is equal to:

$$P(n+1, get(n+1, dt)) \parallel S(n, dt).$$

The induction hypothesis says that $S(n, dt)$ is a solution for Q. So, $S(n + 1, dt)$ is equal to:

$$\begin{aligned}&P(n+1, get(n+1, dt)) \parallel \\ &(\textstyle\sum_{i \in I} \sum_{k : \mathbb{N}} \sum_{e_i : E_i} a_i(f_i(k, get(k, dt), e_i))\, S(n, upd(k, g_i(k, get(k, dt), e_i), dt)) \\ &\qquad \lhd c_i(k, get(k, dt), e_i) \wedge k \leq n \rhd \delta + \\ &\quad \textstyle\sum_{i_1 \in I} \sum_{i_2 \in I} \sum_{k_1 : \mathbb{N}} \sum_{k_2 : \mathbb{N}} \sum_{e_{i_1} : E_{i_1}} \sum_{e_{i_2} : E_{i_2}} \gamma(a_{i_1}, a_{i_2})(f_{i_1}(k_1, get(k_1, dt), e_{i_1})) \\ &\qquad S(n, upd(k_1, g_{i_1}(k_1, get(k_1, dt), e_{i_1}), upd(k_2, g_{i_2}(k_2, get(k_2, dt), e_{i_2}), dt))) \\ &\qquad\quad \lhd c_{i_1}(k_1, get(k_1, dt), e_{i_1}) \wedge c_{i_2}(k_2, get(k_2, dt), e_{i_2}) \wedge \\ &\qquad\quad eq(f_{i_1}(k_1, get(k_1, dt), e_{i_1}), f_{i_2}(k_2, get(k_2, dt), e_{i_2})) \wedge k_1 > k_2 \wedge k_1 \leq n \rhd \delta)\end{aligned}$$

By expanding the parallel operator, and the definition of P where necessary, this term expands to:

$$\begin{aligned}
&\textstyle\sum_{i\in I}\sum_{e_i:E_i} a_i(f_i(n+1,get(n+1,dt),e_i))\\
&\quad (P(n+1,g_i(n+1,get(n+1,dt),e_i))\parallel S(n,dt))\triangleleft c_i(n+1,get(n+1,dt),e_i)\triangleright\delta+\\
&\textstyle\sum_{i\in I}\sum_{k:\mathbb{N}}\sum_{e_i:E_i} a_i(f_i(k,get(k,dt),e_i))\\
&\quad (P(n+1,get(n+1,dt))\parallel\\
&\qquad S(n,upd(k,g_i(k,get(k,dt),e_i),dt)))\triangleleft c_i(k,get(k,dt),e_i)\wedge k\leq n\triangleright\delta+\\
&\textstyle\sum_{i_1\in I}\sum_{i_2\in I}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}}\sum_{e_{i_1}:E_{i_1}}\sum_{e_{i_2}:E_{i_2}}\gamma(a_{i_1},a_{i_2})(f_{i_1}(k_1,get(k_1,dt),e_{i_1}))\\
&\quad (P(n+1,get(n+1,dt))\parallel\\
&\qquad S(n,upd(k_1,g_{i_1}(k_1,get(k_1,dt),e_{i_1}),upd(k_2,g_{i_2}(k_2,get(k_2,dt),e_{i_2}),dt))))\\
&\qquad\quad \triangleleft c_{i_1}(k_1,get(k_1,dt),e_{i_1})\wedge c_{i_2}(k_2,get(k_2,dt),e_{i_2})\wedge\\
&\qquad\quad\ eq(f_i(k_1,get(k_1,dt),e_{i_1}),f_{i_2}(k_2,get(k_2,dt),e_{i_2}))\wedge k_1>k_2\wedge k_1\leq n\triangleright\delta+\\
&\textstyle\sum_{i\in I}\sum_{i'\in I}\sum_{k:\mathbb{N}}\sum_{e_i:E_i}\sum_{e_{i'}:E_{i'}}\gamma(a_i,a_{i'})(f_i(n+1,get(n+1,dt),e_i))\\
&\quad (P(n+1,g_i(n+1,get(n+1,dt),e_i))\parallel S(n,upd(k,g_{i'}(k,get(k,get(k,dt),e_{i'}),dt))))\\
&\qquad \triangleleft c_i(n+1,get(n+1,dt),e_i)\wedge c_{i'}(k,get(k,dt),e_{i'})\wedge k\leq n\wedge\\
&\qquad\ eq(f_i(n+1,get(n+1,dt),e_i),f_{i'}(k,get(k,dt),e_{i'}))\triangleright\delta
\end{aligned}$$

Folding the definition of S with the help of Lemma 3.1 reduces this term to:

$$\begin{aligned}
&\textstyle\sum_{i\in I}\sum_{e_i:E_i} a_i(f_i(n+1,get(n+1,dt),e_i))\\
&\quad S(n+1,upd(n+1,g_i(n+1,get(n+1,dt),e_i),dt))\triangleleft c_i(n+1,get(n+1,dt),e_i)\triangleright\delta+\\
&\textstyle\sum_{i\in I}\sum_{k:\mathbb{N}}\sum_{e_i:E_i} a_i(f_i(k,get(k,dt),e_i))\\
&\quad S(n+1,upd(k,g_i(k,get(k,dt),e_i),dt))\triangleleft c_i(k,get(k,dt),e_i)\wedge k\leq n\triangleright\delta+\\
&\textstyle\sum_{i_1\in I}\sum_{i_2\in I}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}}\sum_{e_{i_1}:E_{i_1}}\sum_{e_{i_2}:E_{i_2}}\gamma(a_{i_1},a_{i_2})(f_{i_1}(k_1,get(k_1,dt),e_{i_1}))\\
&\quad S(n+1,upd(k_1,g_{i_1}(k_1,get(k_1,dt),e_{i_1}),upd(k_2,g_{i_2}(k_2,get(k_2,dt),e_{i_2}),dt)))\\
&\qquad \triangleleft c_{i_1}(k_1,get(k_1,dt),e_{i_1})\wedge c_{i_2}(k_2,get(k_2,dt),e_{i_2})\wedge\\
&\qquad\ eq(f_i(k_1,get(k_1,dt),e_{i_1}),f_{i_2}(k_2,get(k_2,dt),e_{i_2}))\wedge k_1>k_2\wedge k_1\leq n\triangleright\delta+\\
&\textstyle\sum_{i\in I}\sum_{i'\in I}\sum_{k:\mathbb{N}}\sum_{e_i:E_i}\sum_{e_{i'}:E_{i'}}\gamma(a_i,a_{i'})(f_i(n+1,get(n+1,dt),e_i))\\
&\quad S(n+1,upd(n+1,g_i(n+1,get(n+1,dt),e_i),\\
&\qquad upd(k,g_{i'}(k,get(k,get(k,dt),e_{i'}),dt))))\\
&\qquad\quad \triangleleft c_i(n+1,get(n+1,dt),e_i)\wedge c_{i'}(k,get(k,dt),e_{i'})\wedge k\leq n\wedge\\
&\qquad\quad\ eq(f_i(n+1,get(n+1,dt),e_i),f_{i'}(k,get(k,dt),e_{i'}))\triangleright\delta
\end{aligned}$$

By taking the summands pairwise together, this reduces to

$$\begin{aligned}
&\textstyle\sum_{i\in I}\sum_{k:\mathbb{N}}\sum_{e_i:E_i} a_i(f_i(k,get(k,dt),e_i))\,S(n+1,upd(k,g_i(k,get(k,dt),e_i),dt))\\
&\quad \triangleleft c_i(k,get(k,dt),e_i)\wedge k\leq n+1\triangleright\delta+\\
&\textstyle\sum_{i_1\in I}\sum_{i_2\in I}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}}\sum_{e_{i_1}:E_{i_1}}\sum_{e_{i_2}:E_{i_2}}\gamma(a_{i_1},a_{i_2})(f_{i_1}(k_1,get(k_1,dt),e_i))\\
&\quad S(n+1,upd(k_1,g_{i_1}(k_1,get(k_1,dt),e_{i_1}),upd(k_2,g_{i_2}(k_2,get(k_2,dt),e_{i_2}),dt)))\\
&\qquad \triangleleft c_{i_1}(k_1,get(k_1,dt),e_{i_1})\wedge c_{i_2}(k_2,get(k_2,dt),e_{i_2})\wedge\\
&\qquad\ eq(f_1(k_1,get(k_1,dt),e_{i_1}),f_{i_2}(k_2,get(k_2,dt),e_{i_2}))\wedge k_1>k_2\wedge k\leq n+1\triangleright\delta.
\end{aligned}$$

This is the right hand side of (6). So, we can conclude that S is a solution for Q for all n. □

Lemma 3.3. *Equation (4) is convergent.*

Proof. As (2) is convergent, there is a well founded relation $<\subseteq\langle\mathbb{N}\times D\rangle\times\langle\mathbb{N}\times D\rangle$, such that if $d_i(k,d,e_i)$ and $a_i=\tau$, then $\langle k,g_i(k,d,e_i)\rangle<\langle k,d\rangle$.

Using $<$ we can define a wellfounded relation $\prec$ as follows:

$$\langle\langle n_1, dt_1\rangle, \langle n_2, dt_2\rangle\rangle \subseteq \prec \text{ iff } \begin{cases} eq(n_1, n_2), \\ \text{for all } 0 \leq k \leq n_1 : \langle k, get(k, dt_1)\rangle \leq \langle k, get(k, dt_2)\rangle \text{ and} \\ \text{for some } 0 \leq k \leq n_1 : \langle k, get(k, dt_1)\rangle < \langle k, get(k, dt_2)\rangle \end{cases}$$

where $\langle k_1, d_1\rangle \leq \langle k_2, d_2\rangle$ iff $\langle k_1, d_1\rangle < \langle k_2, d_2\rangle$, or $eq(k_1, k_2)$ and $eq(d_1, d_2)$.

Using $\prec$ it is straightforward to see that (4) is convergent. □

Corollary 3.4 (*Parallel Specification Principle*). *Equation (3) has at most one solution for the variable S.*

Proof. Lemma 3.2 says that any solution for S in (3) is a solution for Q in (4). Using Lemma 3.3 CL-RSP expresses that there is at most one solution for Q in (4). Henceforth, equation (3) has at most one solution, too. □

In the following theorem we assume that there is a total reflexive ordering $\leq$ on I. As I is an index set, this is a very reasonable assumption.

Theorem 3.5. *The process S as defined in equation (3) is a (unique) solution for Q in the (convergent) equation below; so $S(n, dt) = Q(n, dt)$ for all $n{:}\mathbb{N}$ and $dt{:}DTable$. The set I and the functions f_i, g_i and c_i are those that occur in equation (2).*

$$\begin{aligned}
&Q(n{:}\mathbb{N}, dt{:}DTable) = \\
&\quad \textstyle\sum_{i\in I} \sum_{k:\mathbb{N}} \sum_{e_i:E_i} a_i(f_i(k, get(k, dt), e_i))\, Q(n, upd(k, g_i(k, get(k, dt), e_i), dt)) \\
&\qquad \lhd c_i(k, get(k, dt), e_i) \wedge k \leq n \rhd \delta + \\
&\quad \textstyle\sum_{i_1\in I} \sum_{i_2\in I \wedge i_2 \leq i_1} \sum_{k_1:\mathbb{N}} \sum_{k_2:\mathbb{N}} \sum_{e_{i_1}:E_{i_1}} \sum_{e_{i_2}:E_{i_2}} \gamma(a_{i_1}, a_{i_2})(f_{i_1}(k_1, get(k_1, dt), e_{i_1})) \\
&\qquad Q(n, upd(k_1, g_{i_1}(k_1, get(k_1, dt), e_{i_1}), upd(k_2, g_{i_2}(k_2, get(k_2, dt), e_{i_2}), dt))) \\
&\qquad\quad \lhd c_{i_1}(k_1, get(k_1, dt), e_{i_1}) \wedge c_{i_2}(k_2, get(k_2, dt), e_{i_2}) \wedge \\
&\qquad\quad eq(f_{i_1}(k_1, get(k_1, dt), e_{i_1}), f_{i_2}(k_2, get(k_2, dt), e_{i_2})) \wedge \\
&\qquad\quad \neg eq(k_1, k_2) \wedge k_1 \leq n \wedge k_2 \leq n \rhd \delta.
\end{aligned}$$

Proof. We show that the right hand sides of equation (4) can straightforwardly be transformed to the right hand side of the equation above. Actually, as the first group of summands of both are equal, we only show that the last group of summands can be transformed. To keep the argument concise we introduce the following two abbreviations:

$$\begin{aligned}
P_{i_1 i_2}(k_1, k_2) = &\textstyle\sum_{e_{i_1}:E_{i_1}} \sum_{e_{i_2}:E_{i_2}} \gamma(a_{i_1}, a_{i_2})(f_{i_1}(k_1, get(k_1, dt), e_{i_1})) \\
&Q(n, upd(k_1, g_{i_1}(k_1, get(k_1, dt), e_{i_1}), upd(k_2, g_{i_2}(k_2, get(k_2, dt), e_{i_2}), dt)))
\end{aligned}$$

$$\begin{aligned}
C_{i_1 i_2}(k_1, k_2) = &\, c_{i_1}(k_1, get(k_1, dt), e_{i_1}) \wedge c_{i_2}(k_2, get(k_2, dt), e_{i_2}) \wedge \\
&eq(f_{i_1}(k_1, get(k_1, dt), e_{i_1}), f_{i_2}(k_2, get(k_2, dt), e_{i_2}))
\end{aligned}$$

A rather trivial but essential observation is that $C_{i_1 i_2}(k_1, k_2) = C_{i_2 i_1}(k_2, k_1)$ and if $\neg eq(k_1, k_2)$ and $eq(f_{i_1}(k_1, get(k_1, dt), e_{i_1}), f_{i_2}(k_2, get(k_2, dt), e_{i_2}))$ then $P_{i_1 i_2}(k_1, k_2) = P_{i_2 i_1}(k_2, k_1)$.

The second group of summands of (4) can now be written as

$$\textstyle\sum_{i_1\in I} \sum_{i_2\in I} \sum_{k_1:\mathbb{N}} \sum_{k_2:\mathbb{N}} P_{i_1 i_2}(k_1, k_2) \lhd C_{i_1 i_2}(k_1, k_2) \wedge k_1 > k_2 \wedge k_1 \leq n \rhd \delta$$

By splitting this summand into $i_2 \leq i_1$ and $i_2 \geq i_1$, splitting $k_1 > k_2$ into $k_1 \geq k_2$ and $\neg eq(k_1, k_2)$ and adding the redundant condition $k_2 \leq n$ this term is equal to:

$$\begin{array}{l}\sum_{i_1\in I}\sum_{i_2\in I\wedge i_2\leq i_1}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}} P_{i_1i_2}(k_1,k_2)\\ \qquad\triangleleft C_{i_1i_2}(k_1,k_2)\wedge k_1\geq k_2\wedge\neg eq(k_1,k_2)\wedge k_1\leq n\wedge k_2\leq n\triangleright\delta+\\ \sum_{i_1\in I}\sum_{i_2\in I\wedge i_2\geq i_1}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}} P_{i_1i_2}(k_1,k_2)\\ \qquad\triangleleft C_{i_1i_2}(k_1,k_2)\wedge k_1\geq k_2\wedge\neg eq(k_1,k_2)\wedge k_1\leq n\wedge k_2\leq n\triangleright\delta\end{array}$$

By changing the order of the summands and by exchanging the names of i_1 and i_2, and k_1 and k_2 in the second group of summands, we obtain:

$$\begin{array}{l}\sum_{i_1\in I}\sum_{i_2\in I\wedge i_2\leq i_1}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}} P_{i_1i_2}(k_1,k_2)\\ \qquad\triangleleft C_{i_1i_2}(k_1,k_2)\wedge k_1\geq k_2\wedge\neg eq(k_1,k_2)\wedge k_1\leq n\wedge k_2\leq n\triangleright\delta+\\ \sum_{i_1\in I}\sum_{i_2\in I\wedge i_2\leq i_1}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}} P_{i_2i_1}(k_2,k_1)\\ \qquad\triangleleft C_{i_2i_1}(k_2,k_1)\wedge k_2\geq k_1\wedge\neg eq(k_2,k_1)\wedge k_2\leq n\wedge k_1\leq n\triangleright\delta\end{array}$$

By applying the 'essential observation' stated above on the second summand we can put i_1, i_2, k_1 and k_2 almost back to their original places:

$$\begin{array}{l}\sum_{i_1\in I}\sum_{i_2\in I\wedge i_2\leq i_1}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}} P_{i_1i_2}(k_1,k_2)\\ \qquad\triangleleft C_{i_1i_2}(k_1,k_2)\wedge k_1\geq k_2\wedge\neg eq(k_1,k_2)\wedge k_1\leq n\wedge k_2\leq n\triangleright\delta+\\ \sum_{i_1\in I}\sum_{i_2\in I\wedge i_2\leq i_1}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}} P_{i_1i_2}(k_1,k_2)\\ \qquad\triangleleft C_{i_1i_2}(k_1,k_2)\wedge k_1\leq k_2\wedge\neg eq(k_1,k_2)\wedge k_1\leq n\wedge k_2\leq n\triangleright\delta\end{array}$$

This term has two similar sets of summands, only differing in that one contains condition $k_1\geq k_2$ and the other contains $k_1\leq k_2$. As either condition must be the case, we can take both groups of summands together obtaining:

$$\begin{array}{l}\sum_{i_1\in I}\sum_{i_2\in I\wedge i_2\leq i_1}\sum_{k_1:\mathbb{N}}\sum_{k_2:\mathbb{N}} P_{i_1i_2}(k_1,k_2)\\ \qquad\triangleleft C_{i_1i_2}(k_1,k_2)\wedge\neg eq(k_1,k_2)\wedge k_1\leq n\wedge k_2\leq n\triangleright\delta\end{array}$$

which is the required right hand side. □

4 Example

We show how to use Theorem 3.5 by expanding Milner's scheduler [13] into linear form. The task of Milner's scheduler is to control n machines such that each machine is alternately switched on and off, and all machines must be switched on in strict order.

A scheduler consists of n cyclers. Cycler i can be described by the equation:

$$\begin{array}{l}C(i,n{:}\mathbb{N}) = \overline{r}(i-_n 1)\,C'(i,n)\\ C'(i,n{:}\mathbb{N}) = s(i)\,(\overline{t}(i)\parallel r(i))\,C(i).\end{array}$$

Here, $\overline{r}(i-_n 1)$ expresses that the cycler receives a signal from neighbour $i-_n 1$ to start; $i-_n 1$ is subtraction modulo $n+1$. Likewise, we use $+_n$ for addition modulo $n+1$. The action $s(i)$ is used to switch machine i on. The action $\overline{t}(i)$ expresses that a termination signal is received from machine i. The action $r(i)$ represents that a signal is sent to the next cycler. The actions $r(i)$ and $\overline{r}(i)$ synchronise to $r^\star(i)$, i.e. $\gamma(r,\overline{r}) = r^\star$.

The cyclers are put in parallel in such a way that the first one is already started.

$$S(k,n{:}\mathbb{N}) = C'(0,n)\triangleleft eq(k,0)\triangleright(S(k-1,n)\parallel C(k,n)).$$

The actual scheduler is given by forbidding r and $\overline{r}$ actions to happen on their own, forcing them to synchronise using the encapsulation operator $\partial_{\{r,\overline{r}\}}$. The resulting synchronisation $r^\star$ is made internal by renaming it to the internal action τ, using the hiding operator $\tau_{\{r^\star\}}$.

$$Sched(n) = \tau_{\{r^\star\}}\partial_{\{r,\overline{r}\}}(S(n,n)).$$

In order to apply theorem 3.5 we must first provide a linear process equation describing a cycler. This is rather straightforward by encoding the states explicitly by natural numbers (cf. [4]). For the equation below it holds that $C(i,n) = P(i,0,n)$ and $C'(i,n) = P(i,1,n)$.

$$\begin{aligned}
&P(i,s,n{:}\mathbb{N}) = \\
&\quad \overline{r}(i -_n 1)\, P(i,1,n) \lhd eq(s,0) \rhd \delta + \\
&\quad s(i)\, P(i,2,n) \lhd eq(s,1) \rhd \delta + \\
&\quad r(i)\, P(i, s +_4 2, n) \lhd eq(s,2) \vee eq(s,3) \rhd \delta + \\
&\quad \overline{t}(i)\, P(i, s +_4 1, n) \lhd eq(s,2) \vee eq(s,4) \rhd \delta.
\end{aligned}$$

We also rewrite the parallel composition, to use the definition of P and to use the variable nt of sort $\mathbb{N}Table$, where $\mathbb{N}Table$ is a table containing natural numbers (cf. $DTable$). If we would have followed the theory in the previous section very precisely, we should have used a table with pairs of natural numbers. As one of these numbers is always n, and already available in Q below, we omit it in the table. So, the new parallel composition is given by:

$$S(k,n{:}\mathbb{N}, nt{:}\mathbb{N}Table) = P(0, get(0,nt), n) \lhd eq(k,0) \rhd (S(k-1,n,nt) \parallel P(k, get(k,nt), n)).$$

If we define the initial table $it : \mathbb{N} \to \mathbb{N}Table$ by ($emnT$ is the empty table of naturals):

$$it(k) = if(k > 0, upd(k, 0, it(k-1)), upd(0,1,emnT))$$

then a simple inductive argument on k shows that $S(k,n,it(k)) = S(k,n)$.

By applying Theorem 3.5 we directly obtain that $S(n,n,nt)$ is characterised by the following equation:

$$\begin{aligned}
&Q(n{:}\mathbb{N}, nt{:}\mathbb{N}Table) = \\
&\quad \textstyle\sum_{k:\mathbb{N}} \overline{r}(k -_n 1)\, Q(n, upd(k,1,nt)) \lhd eq(get(k,nt),0) \wedge k \leq n \rhd \delta + \\
&\quad \textstyle\sum_{k:\mathbb{N}} s(k)\, Q(n, upd(k,2,nt)) \lhd eq(get(k,nt),1) \wedge k \leq n \rhd \delta + \\
&\quad \textstyle\sum_{k:\mathbb{N}} r(k)\, Q(n, upd(k, get(k,nt) +_4 2, nt)) \\
&\qquad \lhd (eq(get(k,nt),2) \vee eq(get(k,nt),3)) \wedge k \leq n \rhd \delta + \\
&\quad \textstyle\sum_{k:\mathbb{N}} \overline{t}(k)\, Q(n, upd(k, get(k,nt) +_4 1, nt)) \\
&\qquad \lhd (eq(get(k,nt),2) \vee eq(get(k,nt),4)) \wedge k \leq n \rhd \delta + \\
&\quad \textstyle\sum_{k_1:\mathbb{N}} \sum_{k_2:\mathbb{N}} r^\star(k_1)\, Q(n, upd(k_1, get(k_1,nt) +_4 2, upd(k_2, 1, nt))) \\
&\qquad \lhd (eq(get(k_1,nt),2) \vee eq(get(k_1,nt),3)) \wedge eq(get(k_2,nt),0) \wedge \\
&\qquad\quad eq(k_1, k_2 -_n 1) \wedge \neg eq(k_1,k_2) \wedge k_1 \leq n \wedge k_2 \leq n \rhd \delta
\end{aligned}$$

So, it follows that $S(n,n) = Q(n, it(n))$.

Now we apply the hiding and encapsulation operator to this equation. We also clean up the last summand by substituting $k_2 := k_1 +_n 1$ and replacing $\neg eq(k_1, k_1 +_n 1)$ by $n > 0$ (in the context of $k_1 \leq n$). We obtain the following equation characterising the behaviour of the scheduler.

$$
\begin{aligned}
&Q'(n{:}\mathbb{N}, nt{:}\mathbb{N}Table) = \\
&\quad \textstyle\sum_{k:\mathbb{N}} s(k)\, Q'(n, upd(k, 2, nt)) \lhd eq(get(k, nt), 1) \wedge k \leq n \rhd \delta + \\
&\quad \textstyle\sum_{k:\mathbb{N}} \bar{t}(k)\, Q'(n, upd(k, get(k, nt) +_4 1, nt)) \\
&\qquad \lhd (eq(get(k, nt), 2) \vee eq(get(k, nt), 4)) \wedge k \leq n \rhd \delta + \\
&\quad \textstyle\sum_{k_1:\mathbb{N}} \tau\, Q'(n, upd(k_1, get(k_1, nt) +_4 2, upd(k_1 +_4 1, 1, nt))) \\
&\qquad \lhd (eq(get(k_1, nt), 2) \vee eq(get(k_1, nt), 3)) \wedge eq(get(k_1 +_4 1, nt), 0) \wedge \\
&\qquad\quad n > 0 \wedge k_1 \leq n \rhd \delta
\end{aligned}
$$

For this equation we have that $Sched(n) = Q'(n, it(n))$.

We do not elaborate on this equation any further. Using the theory in [10] it is rather straightforward to prove correctness, for instance in the form stated in [12].

References

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

[2] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. Grid Protocols based on Synchronous Communication. To appear in *Science of Computer Programming.*

[3] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *Proceedings Concur'94*, Uppsala, Sweden, Lecture Notes in Computer Science no. 836, pages 401-416, Springer Verlag, 1994.

[4] D.J. Bosscher and A. Ponse. Translating a process algebra with symbolic data values to linear format. In U.H. Engberg, K.G. Larsen, and A.S. Skou, editors, *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 119–130. BRICS Notes Series NS-95-2, 1995.

[5] L.-Å. Fredlund, J.F. Groote and H. Korver. Formal Verification of a Leader Election Protocol in Process Algebra. To appear in *Theoretical Computer Science*, 1996?.

[6] J.F. Groote and H. Korver. Correctness proof of the bakery protocol in μCRL. In A. Ponse, C. Verhoef and S.F.M. van Vlijmen, eds, *Algebra of Communicating Processes*, Workshops in Computing, pp. 63–86, 1994.

[7] J.F. Groote and A. Ponse. Proof theory for μCRL: a language for processes with data. In Andrews et al. *Proceedings of the International Workshop on Semantics of Specification Languages.* Workshops in Computing, pages 231–250. Springer Verlag, 1994.

[8] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef and S.F.M. van Vlijmen, eds, *Algebra of Communicating Processes*, Workshops in Computing, pp. 26–62, 1994.

[9] J.F. Groote and J. Springintveld. Verification of a summing protocol. To appear.

[10] J.F Groote and J. Springintveld. Focus Points and Convergent Process Operators. Technical Report 142, Logic Group Preprint Series, Department of Philosophy, Utrecht University, 1995.

[11] H. Korver. Protocol Verification in μCRL. PhD. Thesis. University of Amsterdam. 1994.

[12] H. Korver and J. Springintveld. A Computer-Checked Verification of Milner's Scheduler. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, Sendai, Japan. Lecture Notes in Computer Science 789, Springer-Verlag, 1994.

[13] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.