Analysis of A Leader Election Algorithm in μCRL *

Taolue Chen †
State Key Laboratory of Novel Software Technology,
Nanjing University, Nanjing, Jiangsu, P.R.China 210093
CWI, Department of Software Engineering,
PO Box 94079, 1090 GB Amsterdam, The Netherlands

Tingting Han Jian Lu
State Key Laboratory of Novel Software Technology,
Nanjing University, Nanjing, Jiangsu, P.R.China 210093

Abstract

This paper investigates the applicability of formal methods for the specification and verification of distributed algorithms. The problem of election is an important class of distributed algorithms that are widely studied in the literatures. We prove the correctness of a representative leader election algorithm, that is, the LCR algorithm, developed by LeLann, Chang and Roberts. This algorithm is one of the early election algorithms and serves as a nice benchmark for verification exercises. The verification is based on the μCRL, which is a language for specifying distributed systems and algorithms in an algebraic style and combines the process algebra and (equational) data types. We bring the correctness of the algorithm to a completely formal level. It turns out that this relatively "small" and "simple" algorithm requires a rather involved proof for guaranteeing that it behaves well in all possible circumstance. This paper demonstrates the possibility to deliver completely formal and mechanically verifiable correctness proofs of highly nondeterministic distributed algorithm, which is indispensable in the design and implementation of distributed algorithm and systems.

1 Introduction

It is well known that distributed algorithms are hard to design correctly. This is not only caused by the inherent complexity of distributed systems, but is also due to the lack of adequate techniques to prove the correctness of such algorithms. This means that there are no good ways of validating designs for the distributed system. The current approach to proving correctness of distributed system generally use stylished forms of hand waving that does not always avoid the intricacies and pitfalls that often appear in distributed algorithms [6]. We are convinced that more precise proof techniques need to be applied, which should allow for computer based proof checking. Concretely this means that formal method - a logic based approach should be taken. However, hitherto the applicability of formal methods for the specification and verification of distributed systems is still a much debated issue. For instance, in [4], Chou claims that there are still no formal methods to reason about distributed systems which are both practical and intuitive. Some algorithms, although "small" and "simple" in face, are difficult to prove correct *formally*.

Distributed algorithm is one of the most active areas in the community of parallel and distributed processing. There are many sorts of distributed algorithms. For an excellent survey, we refer the readers to [11]. Among these, leader election is a classical but significant problem. In this problem, a network of identical processes must choose a "leader" from among themselves. The processes are assumed to be indistinguishable, except that they may possess unique *identifiers*. The election problem requires that, in short, starting from a configuration where each process is in the same state, a configuration is reached where exactly one process is in a special state *leader*, while all other processes are in the state *lost*. The difficulty lies in breaking the symmetry. Besides the importance in the theory, the solution for such problem is also of practical applica-

*Funded by NNSFC (60233010, 60273034, 60403014), 863 Program of China (2001AA113110, 2002AA116010), 973 Program of China (2002CB312002).

†Corresponding author. Email: ctl@ics.nju.edu.cn

Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05)
0-7695-2432-X/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

tions. For example, it can be used to implement a fault-tolerant token-passing algorithm; if the token is lost, the leader election algorithm can be invoked to decide which process should possess the token.

In this paper, we study the distributed algorithm, concretely, the leader election algorithm from the prospective of *formal verification*, that is, proving the properties of such algorithms. We consider a representative of early leader election algorithm for unidirectional rings. This problem was first posed by LeLann [10], who also proposed the first solution in the setting of ring networks. In his algorithm, each initiator computes a list of the identities of all initiators, after which the initiator with the largest identity is elected. Each initiator sends a token, containing its identity, via the ring, and this token if forwarded by all processes. It is assumed that the channels are FIFO and an initiator must generate its token before the token of any other initiator is received. When an initiator p receives its own token back, the tokens of all initiators have passed p, and p becomes elected if and only if p is the largest among the initiators. Chang and Roberts [3] improve LeLann's algorithm by removing from the ring all tokens of processes for which it can be seen that they will lose the election. In this paper, following [11], we refer their algorithm as LCR algorithm. In this paper we present a verification of this well-known algorithm in μCRL, a process algebra which allows processes parameterized with data. The correctness of the algorithm is stated as a process equation, the proof of which is a straightforward application of the methodology from [9], a combination of algebraic and assertional techniques. We bring the correctness of the algorithm to a completely formal level. It turns out that this relatively "obvious" algorithm requires a rather involved proof for guaranteeing that it behaves well in all possible circumstance. The results of this paper, to a large extent, demonstrate the possibility to deliver completely formal and mechanically verifiable correctness proofs of highly nondeterministic distributed algorithm, which is indispensable in the design and implementation of distributed algorithm and systems.

The rest of this paper is organized as follows: Some preliminaries are reviewed in the following section. We mainly discuss the LCR algorithm, some background material for μCRL, and *cones and foci* method. In Section 3, we present the whole specification of the algorithm, including the data type and process behavior. Section 4 is devoted to the correctness proof. The paper is concluded with Section 4 where related work is also discussed.

2 Preliminaries

2.1 LCR Algorithm

We assume n processes in a ring topology, connected by unbounded queues. A process can only send messages in a clockwise manner. Initially, each process has a unique identifier (UID, in the following assumed to be a natural number). The task of an algorithm for solving the leader election problem is then to make sure that eventually exactly one process will become the leader. In LCR algorithm, the task that each process in the ring performs is described by the following pseud-code:

var $state_p$
begin
  $state_p$="unknown";
  $value$:=UID;
  send $value$ to $Next_p$;
  while $state_p \neq$ "leader" do
    begin
      receive a message v;
      if $v = value$ **then** $state_p$:="leader"
        else if $v > value$ **then** $value := v$;
    endwhile
end

Note this is a non-formal version. Below we will formalize the processes and their configurations in the ring in μCRL.

2.2 μCRL

μCRL [7] (see also [6]) is a language for specifying distributed systems and algorithms in an algebraic style. It is based on the process algebra ACP [1] extended with equational abstract data types. In a μCRL specification, one part specifies the data types by means of equations $d = e$, while the second part specifies the process behavior. We assume the data sort of booleans *Bool* with constants t and f, and the usual connectives $\vee$, $\wedge$, $\neg$, $\rightarrow$ and $\leftrightarrow$. For a boolean b, we abbreviate $b = \mathbf{t}$ to b and $b = \mathbf{f}$ to $\neg b$.

The data types needed for our μCRL specification of a LCR are presented in Section 3. For the process part of μCRL, the processes are usually represented by process terms, which describe the order in which the actions from a set $\mathcal{A}$ may happen. To each μCRL specification belongs a directed graph, called a labelled transition system, which is defined by the structural operational semantics of μCRL (see [7]). In this labelled transition system, the states are process terms, and the edges are labelled with parameterized actions. Branching bisimulation $\simeq_b$ [12] and standard

IEEE COMPUTER SOCIETY

strong bisimulation $\sim$ are two well-established equivalence relations on the states in labelled transition systems. Conveniently, strong bisimulation equivalence implies branching bisimulation equivalence. The proof theory of μCRL from [7] is sound modulo branching bisimulation equivalence, meaning that if $p = q$ can be derived from it then $p \simeq_b q$. The goal of this paper is to prove that the initial state of the forthcoming μCRL specification of LCR is branching bisimilar to an action which claims the leader. In the proof of this fact, we will use three proof principles from the literature to derive that two μCRL specifications are branching (or even strongly) bisimilar: *sum elimination*, CL-RSP, and *cones and foci*. Due to space restriction, we refer the reader to [5][2] and [9] for details.

The rest of this subsection will be devoted to introducing some important notions. First we introduce the notion of *linear process operator* (LPO in short). LPOs are traditional defined equationally and processes (i.e. linear process equation) can be defined as solutions for LPOs.

Definition 1 *A* linear process operator *(LPO)* Ψ *is an expression of the form*

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

for some finite index set I*, action labels* $a_i \in AL_\tau$*, data types* D_i *and* D_{a_i}*, functions* $f_i : D \to D_i \to D_{a_i}$*,* $g_i : D \to D_i \to D$*, and* $c_i : D \to D_i \to Bool$*.* $\mathbb{P}$ *is the sort of processes.*

We call an LPO *convergent* if the process it defines cannot perform infinite sequences of τ-actions.

Definition 2 *An LPO* Φ *written in Definition 1 is called* convergent *if there is a well-founded ordering* $<$ *on* D *such that for all* $e_\tau : E_\tau, d : D$ *we have that* $b_\tau(d, e_\tau)$ *implies* $g_\tau(d, e_\tau) < d$*.*

Invariants of a system are properties of data that are satisfied throughout the reachable state space of the system.

Definition 3 *An* invariant *of an LPO* Φ *is a function* $\mathcal{I} : D \to Bool$ *such that for all* $a \in Act$*,* $e_a : E_a$ *and* $d : D$ *we have*

$$b_a(d, e_a) \wedge \mathcal{I}(d) \to \mathcal{I}(g_a(d, e_a))$$

3 Formal Description

3.1 Data Types

In this section, the data types used in the μCRL of the LCR are presented.

3.1.1 Booleans

We introduce the data type Bool of booleans.

$$\begin{aligned} &\mathbf{t}, \mathbf{f}: \to \text{Bool} \\ &\vee,\wedge: \text{Bool} \times \text{Bool} \to \text{Bool} \\ &\neg: \text{Bool} \to \text{Bool} \\ &\to,\leftrightarrow: \text{Bool} \times \text{Bool} \to \text{Bool} \end{aligned}$$

$\mathbf{t}$ and $\mathbf{f}$ denote true and false, respectively. The infix operations $\vee$, $\wedge$ represent conjunction and disjunction, respectively. Finally, $\neg$ denotes negation. Then defining equations are:

$$\begin{array}{lcllcl} b \wedge \mathbf{t} & = & b & \neg\mathbf{t} & = & \mathbf{f} \\ b \wedge \mathbf{f} & = & \mathbf{f} & \neg\mathbf{f} & = & \mathbf{t} \\ b \vee \mathbf{t} & = & \mathbf{t} & b \to b' & = & \neg b \vee b' \\ b \wedge \mathbf{f} & = & b & b \leftrightarrow b' & = & (b \to b') \wedge (b' \to b) \end{array}$$

Unless otherwise stated, data parameters in boolean formulas are universally quantified.

3.1.2 If-then-else and Equality

For each data type D in this paper we assume the presence of an operation

$$if : \text{Bool} \times D \times D \to D$$

with as defining equations

$$\begin{aligned} if(\mathbf{t}, d, e) &= d \\ if(\mathbf{f}, d, e) &= e \end{aligned}$$

Furthermore, for each data type D in this paper one can easily define a mapping $eq : D \times D \to \text{Bool}$, such that $eq(d, e)$ holds if and only if $d = e$ can be derived. For notational conventional convenience we take the liberty to write $d = e$ instead of $eq(d, e)$.

3.1.3 Natural Numbers

We introduce the data type $\mathbb{N}$ of natural numbers.

$$\begin{aligned} &0: \to \mathbb{N} \\ &S: \mathbb{N} \to \mathbb{N} \\ &+, -, \cdot: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ &\leq, <, \geq, >: \mathbb{N} \times \mathbb{N} \to \text{Bool} \end{aligned}$$

0 denotes zero and $S(n)$ the successor of n. The infix operations $+, -$, and $\cdot$ represent addition, subtraction and implication, respectively. Finally, the infix operations $\leq, <, \geq$ and $>$ are the less-than (-or-equal) and greater-than (-or-equal) operations. Usually, the sign for multiplication is omitted, and $\neg(i = j)$ is abbreviated to $i \neq j$. Note that we take as binding convention:

$$\begin{aligned} &\{=, \neq\} > \{\cdot\} > \{+, -\} > \\ &\quad \{>, \geq, <, \leq\} > \{\neg\} > \{\wedge, \vee\} > \{\to, \leftrightarrow\} \end{aligned}$$

IEEE COMPUTER SOCIETY

The defining equations are:

$$\begin{array}{lcllcl} i+0 & = & i & 0 \le i & = & t \\ i+S(j) & = & S(i+j) & S(i) \le 0 & = & f \\ i-0 & = & i & S(i) \le S(j) & = & i \le j \\ 0-i & = & 0 & 0 < S(i) & = & t \\ S(i)-S(j) & = & i-j & i < 0 & = & f \\ i \cdot 0 & = & 0 & S(i) < S(j) & = & i < j \\ i \cdot S(j) & = & (i \cdot j)+i0 & i \ge j & = & \neg(j < i) \\ & & & i > j & = & \neg(j \le i) \end{array}$$

3.1.4 Modulo Arithmetic

Since the topology of processes that take part in the leader election is a ring, the calculations modulo n, the number of processes, play an important role. We introduce the following notation for module calculations.

$$\begin{array}{l} | : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ div : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \end{array}$$

$i|_n$ denotes i modulo n, while $i\ div\ n$ denotes i integer divided by n. The modulo operations are defined by the following equations (for $n > 0$).

$$\begin{array}{rcl} i|_n & = & \begin{cases} i & \text{if } i < n \\ (i-n)|_n & \text{o.w.} \end{cases} \\ i\ div\ n & = & \begin{cases} 0 & \text{if } i < n \\ S((i-n)\ div\ n) & \text{o.w.} \end{cases} \end{array}$$

3.2 Specification of LCR

In order to prove the correctness of the algorithm, we must be precise about the behavior of the process. Below we formalize the processes and their configuration in the ring in μCRL.

$$\begin{array}{ll} \textbf{act} & leader : \mathbb{N} \\ & s, r : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ \textbf{proc} & P(k : \mathbb{N}, send : \mathbb{N}) \approx \\ & \quad s(k, (k+1)|_n, send).P(k, send) \\ & \quad + \sum_{v:\mathbb{N}} r((k+n-1)|_n, k, v).(P(k, v) \lhd v > k \rhd \delta \\ & \quad + leader(k) \lhd v = k \rhd \delta) \end{array}$$

Each process of the LCR is modelled by the process $P(k : \mathbb{N}, send : \mathbb{N})$, where parameter k is the identification (UID) of the process, and the parameter *send* is the data the process intend to send each time. The processes of the network interact via matching actions s (send) and r (receive). We state that send actions s communicate with receive actions r by **comm** rule below. For each action, the parameters are source, destination and value.

It remains to connect all processes together, for which we define the parallel composition of m copies of the process P. The result can be viewed as a ring network of processes in the following way, which is represented by the process $Imp(m)$.

$$\begin{array}{ll} \textbf{act} & c : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \\ \textbf{comm} & r|s = c \\ \textbf{proc} & Imp'(m : \mathbb{N}) \approx \\ & \quad (P(m-1, m-1) \| Imp'(m-1)) \lhd m > 0 \rhd P(0,0) \\ & Imp(m : \mathbb{N}) \approx \tau_{\{c\}} \partial_{\{r,s\}} (Imp'(m)) \end{array}$$

Since the algorithm is supposed to select exactly one leader (actually, the one who has the largest UID) after some internal negotiation, we formulate the correctness of the algorithm by the following formula, where "$=$" is to be interpreted as "behaves the same". This is the standard approach in process algebra and the intuitional sense is clear. Note that since there are n processes who join the election, the maximal UID is $n-1$.

Theorem 1 *For $n > 0$, $Imp(n) = leader(n-1)$.*

The remainder of this paper is devoted to proving the above theorem.

4 Proof of the Algorithm

4.1 Linearization

The starting point of our correctness proof is a linear specification in which no parallel operators occur. That is, we intend to describe the leader election algorithm as a μCRL in a state based style, as this is far more convenient for proving purpose.We follow the linearization algorithm presented in [8]. For technical reasons, we first introduce an extra recursive variable X. It follows that

$$\begin{array}{l} P(k : \mathbb{N}, send : \mathbb{N}) \approx \\ \quad s(k, (k+1)|_n, send).P(k, send) \\ \quad + \sum_{v:\mathbb{N}} r((k+n-1)|_n, k, v).X(k, v) \\ X(k : \mathbb{N}, v : \mathbb{N}) \approx \\ \quad P(k, v) \lhd v > k \rhd \delta \\ \quad + leader(k) \lhd v = k \rhd \delta \\ \approx s(k, (k+1)|_n, send).P(k, v) \lhd v > k \rhd \delta \\ \quad + \sum_{v:\mathbb{N}} r((k+n-1)|_n, k, v).X(k, v) \lhd v > k \rhd \delta \\ \quad + leader(k) \lhd v = k \rhd \delta \end{array}$$

Inspection of the processes P and X indicates that there are two different major states between the actions, which is reflected by the newly introduced extra

IEEE COMPUTER SOCIETY

parameter st in the following process definition $Proc$. The states in P are numbered by 1 and those in X get the number 2.

$$\begin{aligned}
&Proc(k:\mathbb{N}, v:\mathbb{N}, st:\mathbb{N}) \approx \\
&\quad s(k,(k+1)|_n, v).Proc(k,v,1) \lhd st = 1 \rhd \delta \\
&\quad + \sum_{v':\mathbb{N}} r((k+n-1)|_n, k, v').Proc(k,v',2) \lhd st = 1 \rhd \delta \\
&\quad + leader(k) \lhd v = k \wedge st = 2 \rhd \delta \\
&\quad + s(k,(k+1)|_n, v).Proc(k,v,1) \lhd v > k \wedge st = 2 \rhd \delta \\
&\quad + \sum_{v':\mathbb{N}} r((k+n-1)|_n, k, v').Proc(k,v',2) \\
&\qquad\qquad \lhd v > k \wedge st = 2 \rhd \delta
\end{aligned}$$

As we have stated in Section 2, invariants of a system can exclude those states that can not be reached from the initial state, thus can compress the state space of the system extensively. To simplify the above presented equation further, we introduce the first invariant, that is:

$$\mathcal{I}_1 : st = 1 \vee st = 2 = \mathbf{t}$$

It follows that $(st = 1) \vee (v > k \wedge st = 2) \leftrightarrow (st = 1) \vee (v > k)$. Thus we can obtain the following simplified definition.

$$\begin{aligned}
&Proc(k:\mathbb{N}, v:\mathbb{N}, st:\mathbb{N}) \approx \\
&\quad s(k,(k+1)|_n, v).Proc(k,v,1) \lhd (st = 1 \vee v > k) \rhd \delta \\
&\quad + \sum_{v':\mathbb{N}} r((k+n-1)|_n, k, v').Proc(k,v',2) \\
&\qquad\qquad \lhd (st = 1 \vee v > k) \rhd \delta \\
&\quad + leader(k) \lhd (v = k \wedge st = 2) \rhd \delta
\end{aligned}$$

The following lemma states the correctness of the linearlization for the single process which joins the election.

Lemma 1 *For any $k \in \mathbb{N}$, $P(k,k) = Proc(k,k,1)$.*

Proof: An easy adaption of [8]. □

In order to define the parallel composition we will use a new sort DTable, which defines the tables indexed by natural numbers, and contains elements of the sort Elt, which is also defined as follows first:

sort	Elt
func	$\langle -,- \rangle : \mathbb{N} \times \mathbb{N} \to \text{Elt}$
	$()_1 : \text{Elt} \to \mathbb{N}$
	$()_2 : \text{Elt} \to \mathbb{N}$
var	$m, n : \mathbb{N}$
rew	$(\langle m,n \rangle)_1 = m$
	$(\langle m,n \rangle)_2 = n$

Note that actually each item of Elt is a pair of two elements, one is used to store the UID of the process while the other is used to store the current state. $\langle -,- \rangle$, $()_1$ and $()_2$ are standard pair functions.

Now, we give the specification of tables, which are simple structures with sufficient functionality for our purpose. The constant emD of sort DTable denotes the empty table. The function upd enters a new data element in the table and the function get gets a specific element from an entry of the table. These operators are characterized by a single equation. We do not specify what happens if an element from the empty table is being read, as we simply do not encounter this situation.

sort	DTable
func	$emD :\to \text{DTable}$
	$upd : \text{DTable} \times \text{Elt} \times \mathbb{N} \to \text{DTable}$
	$get : \text{DTable} \times \mathbb{N} \to \text{Elt}$
var	$m, n : \mathbb{N}$, $e : \text{Elt}$, $dt : \text{DTable}$
rew	$get(upd(dt,e,m),n) = if(m = n, e, get(dt,n))$

In order to define the initial data, we define the table of n entry (indexed by $0, 1, \ldots, n-1$) as follows:

$$\begin{cases} dt(0) & = upd(emD, \langle 0,1 \rangle, 0) \\ dt(m) & = upd(dt(m-1), \langle m,1 \rangle, m) \end{cases}$$

Now, we can give an expansion of Imp, where all operators for parallelism have been removed. The resulting process has the index m and the table bt as parameters. In essence, the complexity of process Imp is now coded using the simple table operations upd and get. In the sequel, let Par act on data of kth component, whose data state is represented by the kth table entry. For convenience, we also let $Par'(m,bt)$ abbreviate $\partial_H(Par(m,bt))$. After expansion and encapsulation it follows that:

$$\begin{aligned}
&Par'(m:\mathbb{N}, bt:\text{DTable}) \approx \\
&\quad \sum_{k:\mathbb{N}} leader(k) \\
&\qquad \lhd k = (get(bt,k))_1 \wedge (get(bt,k))_2 = 2 \wedge k < m \rhd \delta \\
&\quad + \sum_{k_1:\mathbb{N}} \sum_{k_2:\mathbb{N}} c(k_1, (k_1+1)|_n, (get(bt,k_1))_1). \\
&\qquad Par'(m, upd(upd(bt, \langle (get(bt,k_1))_1, 1 \rangle, k_1), \\
&\qquad \langle (get(bt,k_2))_2, 1 \rangle, k_2)) \\
&\qquad\quad \lhd ((get(bt,k_1))_1 > k_1 \vee (get(bt,k_1))_2 = 1) \\
&\qquad\quad \wedge ((get(bt,k_2))_1 > k_2 \vee (get(bt,k_2))_2 = 1) \\
&\qquad\quad \wedge (k_1 = (k_2+n-1)|_n) \wedge ((k_1+1)|_n = k_2) \\
&\qquad\quad \wedge k_1 > k_2 \wedge k_1 < m \rhd \delta
\end{aligned}$$

Note that this linear specification is obtained by stripping all arguments from communication actions, and

IEEE COMPUTER SOCIETY

renaming these actions according to the rule $s|r = c$. We can rewrite the above equation to the following simple one, eliminating all occurrences of $|_n$ from the specification. This is the linear specification $Par'(m : \mathbb{N}, bt : \mathrm{DTable})$ of the LCR, with encapsulation but without hiding, takes the following form.

$$
\begin{aligned}
&Par'(m : \mathbb{N}, bt : \mathrm{DTable}) \approx \\
&\sum_{k=0}^{m-1} leader(k) \lhd k = (get(bt,k))_1 \\
&\qquad \wedge (get(bt,k))_2 = 2 \rhd \delta \\
&+ \sum_{k=0}^{m-2} c(k, k+1, (get(bt,k))_1). \\
&Par'(m, upd(upd(bt, \langle (get(bt,k))_1, 1\rangle, k_1), \\
&\langle (get(bt,k+1))_2, 2\rangle, k+1)) \\
&\lhd((get(bt,k))_1 > k \vee (get(bt,k))_2 = 1) \\
&\wedge((get(bt,k+1))_1 > k+1 \vee (get(bt,k+1))_2 = 1) \rhd \delta \\
&+c(m-1, 0, (get(bt,m-1))_1). \\
&Par'((m, upd(upd(bt, \langle (get(bt,m-1))_1, 1\rangle, k_1), \\
&\langle (get(bt,0))_2, 2\rangle, 0)) \\
&\lhd((get(bt,m-1))_1 > m-1 \vee (get(bt,m-1))_2 = 1) \\
&\wedge((get(bt,0))_1 > 0 \vee (get(bt,0))_2 = 1) \rhd \delta
\end{aligned}
$$

The following lemma states the correctness of the linearlization for the whole processes.

Lemma 2 *For any $n > 0$,*

$$Imp(n) = \tau_{\{c\}}(Par'(n, dt(n-1)))$$

Proof: An easy adaption of [8]. □

Now, the proof of Theorem 1 boils down to proving that for any $n > 0$, $leader(n-1) = \tau_{\{c\}}(Par'(n,dt(n-1)))$, which will be completed in the next section.

4.2 Correctness

The following lemma collects invariants of specification that are needed in the correctness proof. Note the first one rehearses the invariant $\mathcal{I}_1$.

Lemma 3 *The following invariants hold for $\tau_{\{c\}}(Par'(n, dt(0)))$.*

1. $\mathcal{I}_1 \equiv 0 \le k < n \wedge (get(bt,k))_2 = 1 \vee (get(bt,k))_2 = 2$.
2. $\mathcal{I}_2 \equiv 0 \le k < n \wedge (get(bt,k))_1 \le k$.
3. $\mathcal{I}_3 \equiv (get(bt,k))_2 = 1 \rightarrow (get(bt,k))_1 < k$.
4. $\mathcal{I}_4 \equiv (get(bt,k))_2 = 2 \rightarrow (get(bt,k))_1 = k$.

Proof: Directly from the definition. □

Observe the above equation for Par'. The first summand denotes the external actions, that is, if the condition satisfied, one will declare that it is the leader, by some action *leader* with its UID. The second summand is the most complicated. It specifies the internal activity of the table. At each possible action c, the table element in cell i is moved on to cell number $i+1$.

Next, abstraction is applied to the equation, and all actions c are renamed to τ. It is not hard to see that the resulting equation is still convergent; If process $\tau_{\{c\}}(Par')$ is restricted to performing only internal actions (τ-steps), then the process "converges" to a state where no element in the table can move closer to the next cell. This situation is captured by the focus condition.

$$
\begin{aligned}
FC(k, bt) \stackrel{def}{=} &(0 \le k < n \\
&\wedge (get(bt,k))_1 = n-1) \vee (get(bt,k))_2 = 2
\end{aligned}
$$

As pointed out in the previous section, the current goal is to prove the following equation, which states the major conclusion of this section.

$$\tau_{\{c\}}(Par'(n, dt(n-1))) = leader(n-1)$$

Here the cones and foci technique, described in the previous section, can be applied successfully. First, we give the definition for the state mapping h as followings.

First define

$$
\begin{cases}
dtf(0) & = upd(emD, \langle n-1, 2\rangle, 0) \\
dtf(m) & = upd(dtf(m-1), \langle n-1, 2\rangle, m)
\end{cases}
$$

then define state mapping h as $h(m, dt) = (m, dtf(n-1))$. We have already stated the convergence of the equation for Par' after renaming the c_i to τ (so for $\tau_I(Par')$). Considering the renaming matching criteria, we find the following proof obligations:

1. Internal actions in the implementation preserve the mapping. Note that our state mapping h is a constant function, this is clearly true.
2. If the implementation can do a visible action then the specification can do a similar one. That is, formally,

   $$(get(bt,k))_1 = k \vee (get(bt,k))_2 = 2) \rightarrow \mathbf{t}$$

   and

   $$((get(bt,k+1))_1 = k+1) \vee (get(bt,k+1))_2 = 2) \rightarrow \mathbf{t}$$

   These are clearly true.

IEEE COMPUTER SOCIETY

3. If the specification can do a visible action and the focus condition holds, then the implementation can do a similar one.

$$((0 \leq k < n \wedge (get(bt,k))_1 = n-1) \vee (get(bt,k))_2 = 2) \\ \wedge true \rightarrow (k = (get(bt,k))_1 \wedge (get(bt,k))_2 = 2)$$

Note that by $\mathcal{I}_2$ (Lemma 3), $0 \leq k < n \wedge (get(bt,k))_1 \leq k$, thus for any $0 \leq k \leq n-2$, $(get(bt,k))_1 \leq n-2$. Consequently, $n-1 < n$ and $get(bt,n-1))_1 = n-1$ and $(get(bt,n-1))_2 = 2$ implies that $k = (get(bt,k))_1 \wedge (get(bt,k))_2 = 2$. Moreover, by $\mathcal{I}_4$, $(get(bt,k))_2 = 2 \rightarrow (get(bt,k))_1 = k$, thus this also true.

4. The implementation and the specification have the same data parameters on visible actions.

$$k = (get(bt,k))_1 \wedge (get(bt,k))_2 = 2 \rightarrow k = n-1$$

This is an immediate result of Invariant $\mathcal{I}_3$.

5. If the implementation and specification perform a visible action, then the mapping on the (data) state of the process is altered in a similar way. This is definitely trivial, since in our specification and implementation, the visible action has no continue process.

This completes the proofs.

5 Conclusion

In this section, we conclude our work. This paper investigates the applicability of formal methods for the specification and verification of distributed systems. We choose the LCR algorithm, an election algorithm developed by LeLann, Chang and Roberts as the case. We make such a choice because on one hand, the problem of election is a very important class of distributed algorithms that are widely studied in the literatures, on the other hand, such algorithm serves as a nice benchmark for verification exercises. The verification is based on the μCRL, which is a language for specifying distributed systems and algorithms in an algebraic style and combines the process algebra and (equational) data types. We bring the correctness of the algorithm to a completely formal level. It turns out that this relatively "small" and "simple" algorithm requires a rather involved proof for guaranteeing that it behaves well in all possible circumstance. This paper demonstrates the possibility to deliver completely formal and mechanically verifiable correctness proofs of highly nondeterministic distributed algorithm, which we believe is indispensable in the design and implementation of distributed algorithm and systems.

References

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

[2] M. Bezem and J. F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *CONCUR*, volume 836 of *Lecture Notes in Computer Science*, pages 401–416. Springer, 1994.

[3] E. J. H. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283, 1979.

[4] C.-T. Chou. Practical use of the notions of events and causality in reasoning about distributed algorithms. Technical Report 940035, UCLA, 1994.

[5] J. Groote and H. Korver. Correctness proof of the bakery protocol in μCRL. In *Proc. ACP'94, Workshops in computing*, pages 63–86. Springer, 1995.

[6] J. Groote and M.Reniers. Algebraic process verification. In J. Bergstra, A. Ponese, and S. Somlka, editors, *Handbook of Process Algebra*, pages 1151–1208. Elsevier, 2001.

[7] 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, *Semantics of Specification Languages*, Workshops in Computing, pages 232–251. Springer, 1993.

[8] J. F. Groote, A. Ponse, and Y. S. Usenko. Linearization in parallel pCRL. *J. Log. Algebr. Program.*, 48(1-2):39–70, 2001.

[9] J. F. Groote and J. Springintveld. Focus points and convergent process operators: a proof strategy for protocol verification. *J. Log. Algebr. Program.*, 49(1-2):31–60, 2001.

[10] G. L. Lann. Distributed systems - towards a formal approach. In *IFIP Congress*, pages 155–160, 1977.

[11] N. Lynch. *Distributed Algorithm.* Morgan Kaufmann Publishers, 1996.

[12] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

0-7695-2432-X/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY