CWI

Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

A computer-checked verification of Milner's scheduler

H.P. Korver, J. Springintveld

Computer Science/Department of Software Technology

A Computer-Checked Verification of Milner's Scheduler

Henri Korver

CWI

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

`henri@cwi.nl`

Jan Springintveld

Department of Philosophy, Utrecht University

Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

`jans@phil.ruu.nl`

Abstract

We present an equational verification of Milner's scheduler, which we checked by computer. To our knowledge this is the first time that the scheduler is proof-checked for a general number n of scheduled processes.

1   Introduction

The correctness of many protocols crucially depends on the characteristics of data; one can think of the use of natural numbers, modulo calculations, lists, etc. Illustrative examples of such protocols are Milner's Scheduler [17], the Bakery Protocol [9] and the Sliding Window Protocol [20, 21].

However, traditionally process theories do not concentrate on data. For instance, Milner's correctness proof of the scheduler [17] relies for a considerable part on meta-reasoning about data. The presence of informal meta-reasoning obstructs the computer-checked verification of correctness proofs for such protocols. Hence the need arises for a process theory which comprises a formal treatment of data types. μCRL [11, 12, 13], which is process algebra [1] combined with data [5], is such a theory. In addition to the usual process algebra operations, μCRL contains two important constructs relating processes and data: the ($\_ \triangleleft \_ \triangleright \_$) operator (*if-then-else*) and the Σ-operator for summation over data. Moreover processes and the corresponding axioms and rules are parametrised with data and an induction principle for data is added.

As a case study, we formalise the correctness proof of Milner's scheduler in the proof theory of μCRL. The result of this exercise is twofold. First, a bug was detected in Milner's proof, which led to a reformulation of his result: Milner's scheduler only works correct if at least two processes are scheduled. Milner claims that his scheduler also works correctly if only one process is scheduled,

however this is not true in his particular set-up. This is a small flaw, but still! Second, a completely formal and computer-checked proof was obtained. As far as we know this is the first (computer-checked) verification of Milner's scheduler for *every* number $n \geq 2$ of scheduled processes. This is to be contrasted with existing verifications of Milner's scheduler for various *instances* of n by the so-called 'bisimulation tools' (see e.g. [6], where the scheduler is treated for 80 cyclers).

The actual proof checking is done using the system Coq (see [4]), a proof assistant based on type theory. This case study (consisting of giving a formal proof *and* checking it in Coq) is part of a series of such case studies. Protocols that have been verified in this way are the Alternating Bit Protocol [2], a Bounded Retransmission Protocol [10], both in the setting of ACP and μCRL, and the same Bounded Retransmission Protocol in the setting of I/O automata [14].

Among these exercises, the verification of Milner's scheduler stands out, because this protocol has quite a complicated interaction between processes and data. This is reflected in the correctness proof: most proofs in this paper consist of a combination of induction over data types, ordinary process algebra expansions and calculations with sums and conditionals. Hence these proofs are rather intricate; and initially some mistakes were made in the proof that were not easy to repair, all of which were detected while checking the proofs with Coq. This task lasted approximately three months. The complete proof development can be found in the file Scheduler.v, which can be obtained by contacting the authors. The size of this file is about 140 Kbyte. Of this, 20% is taken up by the proofs in section 6, which constitute the core of Milner's proof. Of the remaining 80%, roughly 30% consists of lemmas concerning the data types. The remaining 50% is divided equally over the other sections.

In appendix D, we give a short description of Coq and a small example of a Coq session. For a more detailed exposition of the implementation of process algebra in Coq we refer to [2, 19].

In this paper we concentrate on showing how to formalise specifications and proofs in such a way that their correctness can be verified automatically. Three points deserve mentioning. First, the correctness proof in this paper is given within the so-called branching bisimulation (see [8]). This slightly strengthens Milner's result, since branching bisimulation is a stronger notion of bisimulation than Milner's observation equivalence.

Second, Milner uses the Restriction laws which determine (among other things) the conditions under which the restriction operator may be distributed over the composition operator. Up to now, the proof theory of μCRL did not contain axioms which correspond to these laws and hence this piece of reasoning could not be formalised in μCRL. To remedy this, we have added the so-called *alphabet axioms*. These axioms are still a subject of research: see [15] for a more thorough treatment.

The third point is directly related to the presence of data. Milner uses the restriction operator to rename occurrences of actions that have parameters that vary with the particular state the scheduler is in. The ordinary encapsulation and hiding operators in μCRL are not refined enough for this kind of renaming: either all or none occurrences of an action are renamed to δ (or τ). So we have extended the syntax of μCRL to obtain a finer renaming mechanism for actions.

The paper is organised as follows. In section 3, we present Milner's scheduler and specify it in μCRL. A revised correctness criterion (see above) for Milner's scheduler is formulated in section 4. In section 5, we formalise in μCRL the meta-syntax (the Π-notation) which is the basis of Milner's proof. In section 6, we prove Milner's scheduler correct in μCRL. The proof of Milner is followed as close as possible such that readers who are familiar with it, can concentrate fully on how the proof is made precise in μCRL. A summary of the proof system is given in appendix A. In appendix B, the alphabet axioms are introduced and lemmas that involve these axioms are proved. The data types that are used in the paper are specified in appendix C. Finally, a short introduction to the Coq system is given in appendix D.

As a final remark, we note that, although the results in this paper are all proof-checked, we do not claim that there are no misprints in this paper. Translating formulas from the Coq notation to the usual notation is still a *human* business.

2

2 Acknowledgements

3 Specifying Milner's scheduler

The scheduler as described by Milner [17] schedules n processes $P(i), 1 \leq i \leq n$, in succession modulo n, i.e. after process $P(n)$ process $P(1)$ is activated again. Moreover, a process may never be reactivated before it has terminated. The process $P(i)$ consists of a request for task initiation $\overline{a}(i)$ followed by a (here unspecified) task $Task(i)$ of which termination is indicated by $\overline{b}(i)$.

The scheduler is built from n cyclers which are positioned in a ring as depicted in Figure 1. Cycler
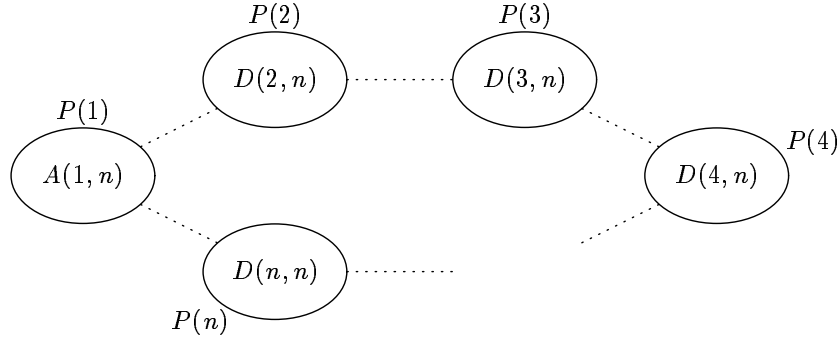

Figure 1: The scheduler.

$A(1, n)$ takes care of process $P(1)$ and cycler $D(i, n), 2 \leq i \leq n$, takes care of process $P(i)$. The first cycler $A(1, n)$ plays a special role as it starts up the system. Cycler $A(i, n)$ initiates process $P(i)$ by performing an action $a(i)$, signaling that $Task(i)$ can start. Then, by performing an action $s(i)$, it informs the next cycler $D(i +_n 1, n)$ that it is $P(i +_n 1)$'s turn to be initiated. Next, it waits for termination of process $P(i)$, indicated by $b(i)$, and in parallel it waits for a signal $s(i -_n 1)$ indicating that it is again $P(i)$'s turn to be initiated. Finally, the cycler returns to its initial state. Cycler $D(i, n)$ first receives a signal indicating that it may start. Then it immediately evolves into the initial state of $A(i, n)$. The formal specification is as follows.

> **act** $\quad a, b, \overline{a}, \overline{b}, \hat{a}, \hat{b}, r, s : nat$
> **comm** $a \mid \overline{a} = \hat{a}, \; b \mid \overline{b} = \hat{b}$
> **proc** $\quad A(i : nat, n : nat) = a(i)s(i)(b(i) \parallel r(i -_n 1))A(i, n)$
> $\qquad D(i : nat, n : nat) = r(i -_n 1)A(i, n)$
> $\qquad P(i : nat) = \overline{a}(i) \, Task(i) \overline{b}(i) P(i)$
> $\qquad Task(i : nat) = \; \ldots$

Here we take the existence of the data type nat (natural numbers) for granted; its specification can be found in appendix C. We also use modulo calculations, e.g. above we have introduced the operator $-_n$ which is subtraction modulo n. Below we shall also use the operator $+_n$ which is addition modulo n. The specification of $-_n$ and $+_n$ can be found in appendix C.

For convenience of reference the following processes are defined.

> **proc** $\quad B(i : nat, n : nat) = b(i)A(i, n)$
> $\qquad E(i : nat, n : nat) = b(i)D(i, n) + r(i -_n 1)B(i, n)$

3

$$C(i : nat, n : nat) = s(i)E(i, n)$$

The whole system is obtained by putting the n cyclers in parallel.

act $\quad c : nat$
comm $r \mid s = c$
proc $\quad \Pi_2(m : nat, n : nat) = (\Pi_2(m - 1, n) \parallel D(m, n)) \lhd m \geq 2 \rhd \delta$
$\qquad Sched(n : nat) = \tau_{\{c\}}(\partial_{\{r,s\}}(A(1, n) \parallel \Pi_2(n, n)))$

Our specification of the scheduler is completely given within the syntax of μCRL. This is in contrast with Milner's CCS specification:

$$Sched \overset{\text{Def}}{=} (A_1 \mid D_2 \mid \ldots \mid D_n) \setminus \{c_1, \ldots, c_n\},$$

where the dots $(\ldots)$ and the variable n (which plays an important role) are informal.

4   A correctness criterion for the scheduler

The system of n cyclers as given above is called a scheduler as the system is supposed to schedule n processes. Below the notion of a scheduler, which is taken from [17], is specified in μCRL.

proc $Schedspec(i : nat, X : list, n : nat) =$
$\qquad \Sigma_{j:nat}(b(j)Schedspec(i, rem(j, X), n) \lhd test(j, X) \rhd \delta) +$
$\qquad \delta \lhd test(i, X) \rhd a(i)Schedspec(i +_n 1, in(i, X), n)$

The process $Schedspec(i, X, n)$ describes a scheduler in the state when any $P(j), j \in X$, may terminate, and also $P(i)$ may be initiated provided that $i \notin X$.

In the specification above we use the function in for inserting an element in a list and the function rem for removing an element from a list. The function $test$ checks whether or not a number is in the list. The specification of in, rem and $test$ can be found in appendix C. Note that we used lists as parameters instead of sets because we found it easier to mechanise the reasoning with lists.

Now, we can formulate the correctness of Milner's scheduler as follows:

$$\boxed{\text{n} \geq 2 \rightarrow Sched(n) = Schedspec(1, \varnothing, n)}$$

One can easily check that the restriction $n \geq 2$ is essential. However, Milner's correctness criterion does not refer to such a restriction, which unavoidably leads to the existence of an incorrect step in the corresponding proof.[1] And this is the only bug we found in Milner's proof; apart from this small oversight his verification is very accurate.

5   Formalising Milner's Π notation

In his proof Milner often uses the meta-notation $\Pi_{i \in X} P_i$ standing for the parallel composition of all processes P_i with $i \in X \subseteq \{1, \ldots, n\}$. In this notation one can rewrite the CCS scheduler given in section 3 as

$$Sched = (A_1 \mid \Pi_{j \in \{2, \ldots, n\}} D_j) \setminus \{c_1, \ldots, c_n\}.$$

By using this notation many crucial steps in Milner's proof are lifted to meta-level. For instance the two following meta-identities (given in CCS notation):

---

[1]In the first step in Subcase $i + 1 \notin X$ (see [17], page 120) the identity $\Pi_{j \notin X \cup \{i\}} D_j = D_{i+1} \mid \Pi_{j \notin X \cup \{i, i+1\}} D_j$ (where $i, i + 1, j \in \{1, \ldots n\}$, $X \subseteq \{1, \ldots n\}$) is used. However this identity is false in case $n < 2$.

1. $i \notin X \rightarrow (\Pi_{j \in X} D_j) \,|\, D_i = \Pi_{j \in X \cup \{i\}} D_j$

2. $i \in X \rightarrow \Pi_{j \in X} D_j = D_i \,|\, (\Pi_{j \in X - \{i\}} D_j)$

are often used in Milner's proof.

Below we formalise Milner's Π-notation in μCRL and prove identities such as given above completely within the proof theory (see Lemma 5.1).

It is straightforward to simulate the set-theoretic operations which are used by Milner by operations on lists. Beside the functions already mentioned, we use the well-known functions 'empty' ($empty$) 'head' (hd) and 'tail' (tl). Now we define the processes Π_D and Π_E as follows.

$$\textbf{proc} \quad \Pi_D(X : list, n : nat) = \delta \triangleleft empty(X) \triangleright (D(hd(X), n) \parallel \Pi_D(tl(X), n))$$
$$\Pi_E(X : list, n : nat) = \delta \triangleleft empty(X) \triangleright (E(hd(X), n) \parallel \Pi_E(tl(X), n))$$

The analogues of the meta-identities mentioned above are given in the following lemma.

Lemma 5.1.

1. $\Pi_D(in(i, X), n) = D(i, n) \parallel \Pi_D(X, n)$

2. $test(i, X) \rightarrow \Pi_D(X, n) = D(i, n) \parallel \Pi_D(rem(i, X), n)$

3. $\Pi_E(in(i, X), n) = E(i, n) \parallel \Pi_E(X, n)$

4. $test(i, X) \rightarrow \Pi_E(X, n) = E(i, n) \parallel \Pi_E(rem(i, X), n)$

Proof.

1.
$$\Pi_D(in(i, X), n)$$
$$= \quad D(hd(in(i, X)), n) \parallel \Pi_D(tl(in(i, X)), n)$$
$$= \quad D(i, n) \parallel \Pi_D(X, n)$$

2. This case is shown with induction on X. The induction follows $\varnothing$ and in.

   - $X = \varnothing : test(i, \varnothing) = F$ and the implication follows.
   - $X = in(j, Y) :$

   $$D(i, n) \parallel \Pi_D(rem(i, in(j, Y)), n)$$
   $$\overset{A.2.1}{=} \quad (D(i, n) \parallel \Pi_D(rem(i, in(j, Y)), n))$$
   $$\triangleleft eq(i, j) \triangleright (D(i, n) \parallel \Pi_D(rem(i, in(j, Y)), n))$$

   $$\overset{C.4.3, C.4.4}{=} \quad (D(j, n) \parallel \Pi_D(Y, n))$$
   $$\triangleleft eq(i, j) \triangleright (D(i, n) \parallel \Pi_D(in(j, rem(i, Y)), n))$$

   $$\overset{5.1.1}{=} \quad \Pi_D(in(j, Y), n)$$
   $$\triangleleft eq(i, j) \triangleright (D(i, n) \parallel D(j, n) \parallel \Pi_D(rem(i, Y), n))$$

   $$\overset{SC}{=} \quad \Pi_D(in(j, Y), n)$$
   $$\triangleleft eq(i, j) \triangleright (D(j, n) \parallel D(i, n) \parallel \Pi_D(rem(i, Y), n))$$

   $$= \quad \Pi_D(in(j, Y), n) \triangleleft eq(i, j) \triangleright (D(j, n) \parallel \Pi_D(Y, n))$$

   by I.H. and C.4.8

5

$$\overset{5.1.1}{=} \quad \Pi_D(in(j, Y), n) \triangleleft eq(i, j) \triangleright \Pi_D(in(j, Y), n)$$

$$\overset{A.2.1}{=} \quad \Pi_D(in(j, Y), n)$$

3. Analogous to (1).

4. Analogous to (2).

$\square$

As a further example of Milner's Π-notation, consider the expression $\Pi_{j \notin X} D_j$, which should be read as $\Pi_{j \in \{1, \ldots, n\} \setminus X} D_j$. In our notation, this becomes: $\Pi_D(X^n, n)$. Here, X^n means $fill(1, n) - X$, where $fill(1, n)$ is the list of natural numbers from 1 up to and including n. For technical convenience, lists are always 'filled' in decreasing order, e.g. $fill(1, 4) = in(4, in(3, in(2, in(1, \varnothing))))$. $X - Y$ is the analogue of set difference and is defined using the function rem. The predicate $X \subseteq Y$ states that every number which occurs in X also occurs in Y. We adopt the convention that we often omit the left hand side of boolean equations for easy notation, i.e. we may write $test(i, X)$ as a short hand for $test(i, X) = T$.

Some care has to be taken to ensure that the representation of sets by lists is well-defined. For instance, $\Pi_{j \in \{1,1\}} D_j = \Pi_{j \in \{1\}} D_j$ but $\Pi_D(in(1, in(1, \varnothing)), n) = D(1, n) \parallel D(1, n) \neq D(1, n) = \Pi_D(in(1, \varnothing), n)$. For ruling this out we only use lists where every element occurs at most once in X. The predicate $unique(X)$ states that X has this property. Another interesting point is the identity $\Pi_{j \in \{1,2\}} D_j = \Pi_{j \in \{2,1\}} D_j$. To deal with this, we define the predicate $perm(X, Y)$ as $X \subseteq Y$ and $Y \subseteq X$. The following lemma shows how the constructions on lists are used for manipulating with the Π_D construct.

Lemma 5.2. *(Π-Permutation).*

1. $\Pi_D(in(i, in(j, X)), n) = \Pi_D(in(j, in(i, X)), n)$

2. $unique(X) \wedge unique(Y) \wedge perm(X, Y) \rightarrow$
   $$\Pi_D(X, n) = \Pi_D(Y, n)$$

3. $test(j, X) \wedge X \subseteq fill(1, n) \wedge unique(X) \rightarrow$
   $$\Pi_D(rem(j, X)^n, n) = \Pi_D(in(j, X^n), n)$$

4. $test(j, X) \wedge \neg eq(i, j) \wedge X \subseteq fill(1, n) \wedge 1 \leq i \wedge i \leq n \wedge unique(X) \rightarrow$
   $$\Pi_D(in(i, rem(j, X))^n, n) = \Pi_D(in(j, in(i, X)^n), n)$$

Proof.

1. $\Pi_D(in(i, in(j, X)), n)$
   $$\overset{5.1.1}{=} \quad D(i, n) \parallel D(j, n) \parallel \Pi_D(X, n)$$
   $$\overset{SC}{=} \quad D(j, n) \parallel D(i, n) \parallel \Pi_D(X, n)$$
   $$\overset{5.1.1}{=} \quad \Pi_D(in(j, in(i, X)), n)$$

2. The key step is 5.2.1.

3. By 5.2.2 and the fact that $perm(rem(j, X)^n, in(j, X^n))$, $unique(rem(j, X)^n)$ and $unique(in(j, X^n))$.

4. By 5.2.2 and the fact that $perm(in(i, rem(j, X))^n, in(j, in(i, X)^n))$, $unique(in(i, rem(j, X))^n)$ and $unique(in(j, in(i, X)^n))$.

$\square$

Lemma 5.2.2 states that lists behave like sets when they appear as parameter in Π. In the next lemma it is shown how we can expand the Π-construct to a summation. This is one of the key steps in the main proof.

Lemma 5.3. *(Π-Expansion).*

1. $unique(X) \rightarrow$
$$\Pi_D(X, n) = \Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, X), n)) \triangleleft test(j, X) \triangleright \delta)$$

2. $unique(X) \rightarrow$
$$\Pi_E(X, n) = \Sigma_{j:nat}(b(j)(D(j, n) \parallel \Pi_E(rem(j, X), n)) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \Sigma_{j:nat}(r(j -_n 1)(B(j, n) \parallel \Pi_E(rem(j, X), n)) \triangleleft test(j, X) \triangleright \delta)$$

For proving this lemma we use the following auxiliary proposition.

Proposition 5.4.

$$\Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, in(i, X)), n)) \triangleleft test(j, in(i, X)) \triangleright \delta) =$$
$$\Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, in(i, X)), n)) \triangleleft test(j, X) \triangleright \delta)$$
$$+ r(i -_n 1)(A(i, n) \parallel \Pi_D(X, n))$$

Proof.

$$\Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, in(i, X)), n)) \triangleleft test(j, in(i, X)) \triangleright \delta)$$

$\overset{C.4.1}{=}$ $\quad \Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, in(i, X)), n)) \triangleleft test(j, X) \text{ or } eq(i, j) \triangleright \delta)$

$= \quad \Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, in(i, X)), n)) \triangleleft test(j, X) \triangleright \delta)$
$+ \quad \Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, in(i, X)), n)) \triangleleft eq(i, j) \triangleright \delta)$

by C.1.2 followed by SUM4

$= \quad \Sigma_{j:nat}(r(j -_n 1)(A(j, n) \parallel \Pi_D(rem(j, in(i, X)), n)) \triangleleft test(j, X) \triangleright \delta)$
$+ \quad r(i -_n 1)(A(i, n) \parallel \Pi_D(X, n))$

by Sum Elimination (Lemma A.3) followed by Lemma C.4.4

$\square$

Now we can proceed with the proof of the lemma given above.

Proof of Lemma 5.3.

1. With induction on X.

   - $X = \varnothing$:

7

$$\Pi_D(\varnothing, n) = \delta = \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(rem(j,\varnothing),n)) \triangleleft test(j,\varnothing) \triangleright \delta)$$

- $X = in(i,Y):$

  $\Pi_D(in(i,Y),n)$
  $$\overset{5.1.1}{=} \quad D(i,n) \parallel \Pi_D(Y,n)$$

  $$\overset{CM1}{=} \quad \Pi_D(Y,n) \parallel\!\!\!\perp D(i,n) + D(i,n) \parallel\!\!\!\perp \Pi_D(Y,n) + D(i,n) \mid \Pi_D(Y,n)$$

  $$\overset{I.H.(twice)}{=} \quad \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(rem(j,Y),n)) \triangleleft test(j,Y) \triangleright \delta) \parallel\!\!\!\perp D(i,n)$$
  $$+ \quad D(i,n) \parallel\!\!\!\perp \Pi_D(Y,n)$$
  $$+ \quad D(i,n) \mid \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(rem(j,Y),n)) \triangleleft test(j,Y) \triangleright \delta)$$

  $$= \quad \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(rem(j,Y),n) \parallel D(i,n)) \triangleleft test(j,Y) \triangleright \delta)$$
  $$+ \quad D(i,n) \parallel\!\!\!\perp \Pi_D(Y,n)$$

  by Sum Expansion (Lemma A.4)

  $$\overset{5.1.1}{=} \quad \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(in(i,rem(j,Y)),n)) \triangleleft test(j,Y) \triangleright \delta)$$
  $$+ \quad D(i,n) \parallel\!\!\!\perp \Pi_D(Y,n)$$

  $$\overset{C.4.3}{=} \quad \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(rem(j,in(i,Y)),n)) \triangleleft test(j,Y) \triangleright \delta)$$
  $$+ \quad D(i,n) \parallel\!\!\!\perp \Pi_D(Y,n)$$

  the application of C.4.3 hangs on $unique(in(i,Y)) \wedge test(j,Y) \rightarrow \neg eq(i,j)$

  $$\overset{CM3}{=} \quad \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(rem(j,in(i,Y)),n)) \triangleleft test(j,Y) \triangleright \delta)$$
  $$+ \quad r(i -_n 1)(A(i,n) \parallel \Pi_D(Y,n))$$

  $$\overset{5.4}{=} \quad \Sigma_{j:nat}(r(j -_n 1)(A(j,n) \parallel \Pi_D(rem(j,in(i,Y)),n)) \triangleleft test(j,in(i,Y)) \triangleright \delta)$$

2. Similar to (1).

$\square$

The next lemma states that we can simulate the Π_2 process from the specification by the Π_D and the *fill*-construct. This will be used in the main proof in section 6.

Lemma 5.5.

1. $m \geq 2 \rightarrow \Pi_2(m,n) \parallel D(S(m),n) = \Pi_2(S(m),n)$

2. $m \geq 2 \rightarrow \Pi_D(fill(2,m),n) = \Pi_2(m,n)$

Proof.

1. $\Pi_2(S(m),n) = \Pi_2(S(m)-1,n) \parallel D(S(m),n) = \Pi_2(m,n) \parallel D(S(m),n)$

2. We prove that $\Pi_D(fill(2,k+2),n) = \Pi_2(k+2,n)$ by induction on k.

   - If $k=0$ then

$$\Pi_D(\mathit{fill}(2,2),n)$$
$$= \qquad \Pi_2(in(2,\varnothing),n) \stackrel{5.1.1}{=} D(2,n) \parallel \Pi_D(\varnothing,n) = D(2,n) \parallel \delta \stackrel{\text{SC}}{=} \Pi_2(2,n)$$

- If $k = S(k')$ then

$$\Pi_D(\mathit{fill}(2,S(k'+2)),n)$$
$$\stackrel{\text{C.4.5}}{=} \quad \Pi_D(in(S(k'+2),\mathit{fill}(2,k'+2)),n)$$
$$\stackrel{5.1.1}{=} \quad D(S(k'+2),n) \parallel \Pi_D(\mathit{fill}(2,k'+2),n)$$
$$\stackrel{\text{I.H}}{=} \quad D(S(k'+2),n) \parallel \Pi_2(k'+2,n)$$
$$\stackrel{\text{SC}}{=} \quad \Pi_2(k'+2,n) \parallel D(S(k'+2),n)$$
$$\stackrel{5.5.1}{=} \quad \Pi_2(S(k'+2),n)$$

$\square$

6 The correctness proof

In this section we verify that Milner's scheduler indeed satisfies the criterion stated in section 4. This is proved as Theorem 6.2.5. The essential step in Milner's proof is the introduction of the process

$$\begin{aligned}
\textbf{proc} \quad & Sched(i:nat, X:list, n:nat) = \\
& \tau_{\{c\}}(\partial_{\{r,s\}}(\\
& \qquad\qquad (B(i,n) \parallel \Pi_D(X^n,n) \parallel \Pi_E(rem(i,X),n)) \\
& \qquad\qquad \triangleleft test(i,X) \triangleright \\
& \qquad\qquad (A(i,n) \parallel \Pi_D(in(i,X)^n,n) \parallel \Pi_E(X,n))))
\end{aligned}$$

which forms the bridge between the processes $Sched(n)$ and $Schedspec(i,X,n)$. We follow Milner's proof very closely. First we show that the process $Sched(i,X,n)$ satisfies the (guarded) defining equation of $Schedspec(i,X,n)$. This is done by distinguishing two cases: the case where X contains number i and the case where X does not. Then by using RSP we have $Sched(i,X,n) = Schedspec(i,X,n)$. Finally, a simple calculation shows that $Sched(n)$ is an instance of $Sched(i,X,n)$, i.e. $Sched(n) = Schedspec(1,\varnothing,n)$, and we are done. All these calculations can be found in Theorem 6.2, the main proof.

The main proof relies on two important steps which we treat below. The scheduler can be in two states where it is unstable, i.e. it can perform one or more invisible τ-actions. Milner shows that these states are equivalent to a stable state.

This involves renaming actions that contain data parameters. Due to the presence of data parameters in actions we have to extend the syntax of the hiding and encapsulation operators. Recall that $\partial_{\{r\}}(p)$ is the result of renaming *all* occurrences of r in p to δ. But sometimes we want to rename only those occurrences of r in p of the form $r(i)$ and leave occurrences of the form $r(i +_n 1)$ unchanged. See the proof of Proposition B.1.1 for an example. The same holds for the τ-operator. So we extend the syntax in such a way that we can write $\partial_{<r(i)>}(p)$ (and $\tau_{<r(i)>}(p)$) and give axioms stating that these operators have the desired properties. These can be found in appendix B. A full treatment of the new syntax and axioms is given in [15].

Another feature of the proof below is that it can already be given within branching bisimulation which is less-identifying than Milner's weak bisimulation.

Lemma 6.1. *For easy notation we write '$\mathcal{F}(p)$' for '$\tau_{<c(i)>}(\partial_{<s(i),r(i)>}(p))$', where p is an arbitrary μCRL process.*

9

1. $\tau\mathcal{F}(C(i,n) \parallel D(i +_n 1, n)) = \tau\mathcal{F}(E(i,n) \parallel A(i +_n 1, n))$

2. $\tau\mathcal{F}(C(i,n) \parallel E(i +_n 1, n)) = \tau\mathcal{F}(E(i,n) \parallel B(i +_n 1, n))$

Proof.

1.
$$\tau\mathcal{F}(C(i,n) \parallel D(i +_n 1, n))$$

$$\overset{\text{expansion}}{=} \quad \tau\mathcal{F}((s(i) \,|\, r((i +_n 1) -_n 1))(E(i,n) \parallel A(i +_n 1, n)))$$

$$= \quad \tau\mathcal{F}(c(i)(E(i,n) \parallel A(i +_n 1, n)))$$

by CF1$'$ in combination with Lemma C.3.2

$$\overset{\text{TA,DA}}{=} \quad \tau\tau\mathcal{F}(E(i,n) \parallel A(i +_n 1, n))$$

$$\overset{\text{B1}}{=} \quad \tau\mathcal{F}(E(i,n) \parallel A(i +_n 1, n))$$

2.
$$\tau\mathcal{F}(C(i,n) \parallel E(i +_n 1, n))$$

$$\overset{\text{expansion}}{=} \quad \tau\mathcal{F}(b(i +_n 1)(C(i,n) \parallel D(i +_n 1, n))$$
$$\qquad + (s(i) \,|\, r((i +_n 1) -_n 1))(E(i,n) \parallel B(i +_n 1, n)))$$

$$= \quad \tau\mathcal{F}(b(i +_n 1)\tau(C(i,n) \parallel D(i +_n 1, n))$$
$$\qquad + \tau(E(i,n) \parallel B(i +_n 1, n)))$$

by B1 and similar computations as in 6.1.1 for reducing the communication result to τ

$$\overset{\text{6.1.1,B1}}{=} \quad \tau\mathcal{F}(\; b(i +_n 1)(C(i,n) \parallel A(i +_n 1, n))$$
$$\qquad + \tau(E(i,n) \parallel B(i +_n 1, n)) \;)$$

$$\overset{\text{partial exp.}}{=} \quad \tau\mathcal{F}(\; b(i +_n 1)(C(i,n) \parallel A(i +_n 1, n))$$
$$\qquad + \tau(b(i +_n 1)(C(i,n) \parallel A(i +_n 1, n)) + E(i,n) \parallel B(i +_n 1, n))$$
$$\qquad)$$

$$\overset{\text{TA,DA}}{=} \quad \tau(\; \mathcal{F}(b(i +_n 1)(C(i,n) \parallel A(i +_n 1, n)))$$
$$\qquad + \tau(\mathcal{F}(b(i +_n 1)(C(i,n) \parallel A(i +_n 1, n))) + \mathcal{F}(E(i,n) \parallel B(i +_n 1, n)))$$
$$\qquad)$$

$$\overset{\text{B2}}{=} \quad \tau(\; \mathcal{F}(b(i +_n 1)(C(i,n) \parallel A(i +_n 1, n)))$$
$$\qquad + \mathcal{F}(E(i,n) \parallel B(i +_n 1, n)))$$

$$\overset{\text{TA,DA}}{=} \quad \tau\mathcal{F}(\; b(i +_n 1)(E(i,n) \parallel A(i +_n 1, n))$$
$$\qquad + E(i,n) \parallel B(i +_n 1, n)$$
$$\qquad)$$

$$\overset{\text{partial exp.}}{=} \quad \tau\mathcal{F}(E(i,n) \parallel B(i +_n 1, n))$$

$\square$

Now we have reached the point where we can prove the main theorem.

Theorem 6.2. *We write 'Cond' for '$n \geq 2 \wedge i \geq 1 \wedge i \leq n \wedge X \subseteq \mathit{fill}(1, n) \wedge \mathit{unique}(X)$'.*

1. $\mathit{test}(i, X) \wedge \mathit{Cond} \rightarrow \ \mathit{Sched}(i, X, n) = \Sigma_{j:nat}(b(j)\mathit{Sched}(i, \mathit{rem}(j, X), n) \triangleleft \mathit{test}(j, X) \triangleright \delta)$

2. $\neg\mathit{test}(i, X) \wedge \mathit{Cond} \rightarrow \ \mathit{Sched}(i, X, n) = \Sigma_{j:nat}(b(j)\mathit{Sched}(i, \mathit{rem}(j, X), n) \triangleleft \mathit{test}(j, X) \triangleright \delta) +$
$a(i)\mathit{Sched}(i +_n 1, \mathit{in}(i, X), n)$

3. $\mathit{Cond} \rightarrow \ \mathit{Sched}(i, X, n) = \mathit{Schedspec}(i, X, n)$

4. $n \geq 2 \rightarrow \mathit{Sched}(n) = \mathit{Sched}(1, \varnothing, n)$

5. $n \geq 2 \rightarrow \mathit{Sched}(n) = \mathit{Schedspec}(1, \varnothing, n)$.

In (1) we may replace $n \geq 2$ in *Cond* by $n \geq 1$.

Proof.

1. $\quad \mathit{Sched}(i, X, n)$

$= \quad \tau_{\{c\}}(\partial_{\{r,s\}}(B(i, n) \parallel \Pi_D(X^n, n) \parallel \Pi_E(\mathit{rem}(i, X), n)))$

$= \quad b(i)\tau_{\{c\}}(\partial_{\{r,s\}}(A(i, n) \parallel \Pi_D(X^n, n) \parallel \Pi_E(\mathit{rem}(i, X), n)))$
$+ \quad \Sigma_{j:nat}(b(j)$
$\quad\quad \tau_{\{c\}}(\partial_{\{r,s\}}(B(i, n) \parallel \Pi_D(X^n, n) \parallel D(j, n) \parallel$
$\quad\quad\quad \Pi_E(\mathit{rem}(j, \mathit{rem}(i, X)), n))) \triangleleft \mathit{test}(j, \mathit{rem}(i, X)) \triangleright \delta)$

by expansion, using Π-Expansion (5.3) and Sum Expansion (A.4)

$\overset{\text{C.4.12}}{=} \quad b(i)\mathit{Sched}(i, \mathit{rem}(i, X), n)$
$+ \quad \Sigma_{j:nat}(b(j)$
$\quad\quad \tau_{\{c\}}(\partial_{\{r,s\}}(B(i, n) \parallel \Pi_D(X^n, n) \parallel D(j, n) \parallel$
$\quad\quad\quad \Pi_E(\mathit{rem}(j, \mathit{rem}(i, X)), n))) \triangleleft \mathit{test}(j, \mathit{rem}(i, X)) \triangleright \delta)$

$\overset{\text{5.1.1}}{=} \quad b(i)\mathit{Sched}(i, \mathit{rem}(i, X), n)$
$+ \quad \Sigma_{j:nat}(b(j)$
$\quad\quad \tau_{\{c\}}(\partial_{\{r,s\}}(B(i, n) \parallel \Pi_D(\mathit{in}(j, X^n), n) \parallel$
$\quad\quad\quad \Pi_E(\mathit{rem}(j, \mathit{rem}(i, X)), n))) \triangleleft \mathit{test}(j, \mathit{rem}(i, X)) \triangleright \delta)$

$\overset{\text{5.2.3,C.4.9}}{=} \quad b(i)\mathit{Sched}(i, \mathit{rem}(i, X), n)$
$+ \quad \Sigma_{j:nat}(b(j)\mathit{Sched}(i, \mathit{rem}(j, X), n) \triangleleft \mathit{test}(j, \mathit{rem}(i, X)) \triangleright \delta)$

$\overset{\text{A.5}}{=} \quad \Sigma_{j:nat}(b(j)\mathit{Sched}(i, \mathit{rem}(j, X), n) \triangleleft \mathit{test}(j, X) \triangleright \delta)$

2. $\quad \mathit{Sched}(i, X, n)$

$= \quad \tau_{\{c\}}(\partial_{\{r,s\}}(A(i, n) \parallel \Pi_D(\mathit{in}(i, X)^n, n) \parallel \Pi_E(X, n)))$

$= \quad \Sigma_{j:nat}(b(j)$
$\quad\quad \tau_{\{c\}}(\partial_{\{r,s\}}(A(i, n) \parallel \Pi_D(\mathit{in}(i, X)^n, n) \parallel D(j, n) \parallel$
$\quad\quad\quad \Pi_E(\mathit{rem}(j, X), n))) \triangleleft \mathit{test}(j, X) \triangleright \delta)$
$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(C(i, n) \parallel \Pi_D(\mathit{in}(i, X)^n, n) \parallel \Pi_E(X, n)))$

11

by expansion, using Π-Expansion (5.3) and Sum Expansion (A.4)

$$\overset{5.1.1,5.2.4}{=} \quad \Sigma_{j:nat}(b(j)Sched(i, rem(j, X), n) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(C(i, n) \parallel \Pi_D(in(i, X)^n, n) \parallel \Pi_E(X, n)))$$

$$\overset{A.2.1}{=} \quad \Sigma_{j:nat}(b(j)Sched(i, rem(j, X), n) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(C(i, n) \parallel \Pi_D(in(i, X)^n, n) \parallel \Pi_E(X, n)))$$
$$\triangleleft test(i +_n 1, X)\triangleright$$
$$a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(C(i, n) \parallel \Pi_D(in(i, X)^n, n) \parallel \Pi_E(X, n)))$$

$$\overset{5.1.2,5.1.4}{=} \quad \Sigma_{j:nat}(b(j)Sched(i, rem(j, X), n) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(C(i, n) \parallel E(i +_n 1, n) \parallel$$
$$\Pi_D(in(i, X)^n, n) \parallel \Pi_E(rem(i +_n 1, X), n)))$$
$$\triangleleft test(i +_n 1, X)\triangleright$$
$$a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(C(i, n) \parallel D(i +_n 1, n) \parallel$$
$$\Pi_D(rem(i +_n 1, in(i, X)^n), n) \parallel \Pi_E(X, n)))$$

in using 5.1.2,5.1.4 above we need $test(i +_n 1, in(i, X)^n)$ which holds because $\neg test(i +_n 1, X)$

$$\overset{B.3.1,B.3.2}{=} \quad \Sigma_{j:nat}(b(j)Sched(i, rem(j, X), n) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(\mathcal{F}(C(i, n) \parallel E(i +_n 1, n)) \parallel$$
$$\Pi_D(in(i, X)^n, n) \parallel \Pi_E(rem(i +_n 1, X), n)))$$
$$\triangleleft test(i +_n 1, X)\triangleright$$
$$a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(\mathcal{F}(C(i, n) \parallel D(i +_n 1, n)) \parallel$$
$$\Pi_D(rem(i +_n 1, in(i, X)^n), n) \parallel \Pi_E(X, n)))$$

$$\overset{A.7\,(twice)}{=} \quad \Sigma_{j:nat}(b(j)Sched(i, rem(j, X), n) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(\tau\mathcal{F}(C(i, n) \parallel E(i +_n 1, n)) \parallel$$
$$\Pi_D(in(i, X)^n, n) \parallel \Pi_E(rem(i +_n 1, X), n)))$$
$$\triangleleft test(i +_n 1, X)\triangleright$$
$$a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(\tau\mathcal{F}(C(i, n) \parallel D(i +_n 1, n)) \parallel$$
$$\Pi_D(rem(i +_n 1, in(i, X)^n), n) \parallel \Pi_E(X, n)))$$

$$\overset{6.1.2,6.1.1}{=} \quad \Sigma_{j:nat}(b(j)Sched(i, rem(j, X), n) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(\tau\mathcal{F}(E(i, n) \parallel B(i +_n 1, n)) \parallel$$
$$\Pi_D(in(i, X)^n, n) \parallel \Pi_E(rem(i +_n 1, X), n)))$$
$$\triangleleft test(i +_n 1, X)\triangleright$$
$$a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(\tau\mathcal{F}(E(i, n) \parallel A(i +_n 1, n)) \parallel$$
$$\Pi_D(rem(i +_n 1, in(i, X)^n), n) \parallel \Pi_E(X, n)))$$

$$= \quad \Sigma_{j:nat}(b(j)Sched(i, rem(j, X), n) \triangleleft test(j, X) \triangleright \delta)$$
$$+ \quad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(E(i, n) \parallel B(i +_n 1, n) \parallel$$
$$\Pi_D(in(i, X)^n, n) \parallel \Pi_E(rem(i +_n 1, X), n)))$$
$$\triangleleft test(i +_n 1, X)\triangleright$$
$$a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(E(i, n) \parallel A(i +_n 1, n) \parallel$$
$$\Pi_D(rem(i +_n 1, in(i, X)^n), n) \parallel \Pi_E(X, n)))$$

by applying the 6th and the 7th step in reverse direction

$$\overset{5.1.3 \ \text{(twice)}}{=} \quad \Sigma_{j:nat}(b(j)Sched(i,rem(j,X),n) \triangleleft test(j,X) \triangleright \delta)$$
$$+ \qquad a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(B(i+_n 1,n) \ \| $$
$$\Pi_D(in(i,X)^n,n) \ \| \ \Pi_E(in(i,rem(i+_n 1,X)),n)))$$
$$\triangleleft test(i+_n 1,X) \triangleright$$
$$a(i)\tau_{\{c\}}(\partial_{\{r,s\}}(A(i+_n 1,n) \ \| $$
$$\Pi_D(rem(i+_n 1,in(i,X)^n),n) \ \| \ \Pi_E(in(i,X),n)))$$

$$\overset{C.4.11,C.4.10}{=} \quad \Sigma_{j:nat}(b(j)Sched(i,rem(j,X),n) \triangleleft test(j,X) \triangleright \delta)$$
$$+ \qquad a(i)Sched(i+_n 1,in(i,X),n)$$
$$\triangleleft test(i+_n 1,X) \triangleright$$
$$a(i)Sched(i+_n 1,in(i,X),n)$$

$$\overset{A.2.1}{=} \qquad \Sigma_{j:nat}(b(j)Sched(i,rem(j,X),n) \triangleleft test(j,X) \triangleright \delta)$$
$$+ \qquad a(i)Sched(i+_n 1,in(i,X),n)$$

3. Define the (guarded) system:

> **func** $Cond : nat \times list \times nat \rightarrow$ **Bool**
> **rew** $Cond(i,X,n) = n \geq 2$ and $i \geq 1$ and $i \leq n$ and $X \subseteq fill(1,n)$ and $unique(X)$
> **proc** $G(i : nat, X : list, n : nat) =$
> $$(\Sigma_{j:nat}(b(j)G(i,rem(j,X),n) \triangleleft test(j,X) \triangleright \delta) +$$
> $$\delta \triangleleft test(i,X) \triangleright a(i)G(i+_n 1,in(i,X),n))$$
> $$\triangleleft Cond(i,X,n) \triangleright \delta$$

We claim that both expressions $\lambda iXn.Schedspec(i,X,n) \triangleleft Cond(i,X,n) \triangleright \delta$ and $\lambda iXn.Sched(i,X,n) \triangleleft Cond(i,X,n) \triangleright \delta$ are solutions for G.[2] It is straightforward to see that the first expression is a solution for G. For the other we have to show that the following equation is derivable:

$$Sched(i,X,n) \triangleleft Cond(i,X,n) \triangleright \delta \ =$$
$$(\Sigma_{j:nat}(b(j)(Sched(i,rem(j,X),n) \triangleleft Cond(i,rem(j,X),n) \triangleright \delta) \triangleleft test(j,X) \triangleright \delta) +$$
$$\delta \triangleleft test(i,X) \triangleright a(i)(Sched(i+_n 1,in(i,X),n) \triangleleft Cond(i+_n 1,in(i,X),n) \triangleright \delta))$$
$$\triangleleft Cond(i,X,n) \triangleright \delta$$

We abbreviate this equation by Φ. For showing that Φ holds, we only have to distinguish two cases:

(I) $test(i,X) \rightarrow \Phi$,

(II) $\neg test(i,X) \rightarrow \Phi$.

Now it is easy to see that (I) and (II) are equivalent with the formulas stated in resp. Theorem 6.2.1 and Theorem 6.2.2. Therefore we know that Φ holds and therewith $\lambda iXn.Sched(i,X,n) \triangleleft Cond(i,X,n) \triangleright \delta$ is a solution for G. By RSP we then have

$$Sched(i,X,n) \triangleleft Cond(i,X,n) \triangleright \delta = Schedspec(i,X,n) \triangleleft Cond(i,X,n) \triangleright \delta$$

---

[2]To be able to substitute data in parametrised processes we use λ-notation in the obvious way (see [12]).

from which we can easily derive

$$Cond \rightarrow Sched(i, X, n) = Schedspec(i, X, n)$$

which finishes the proof.

4. $Sched(1, \varnothing, n)$

$$= \tau_{\{c\}}(\partial_{\{r,s\}}(A(1,n) \parallel \Pi_D(in(1,\varnothing)^n, n) \parallel \Pi_E(\varnothing, n)))$$
$$\stackrel{C.4.2}{=} \tau_{\{c\}}(\partial_{\{r,s\}}(A(1,n) \parallel \Pi_D(fill(2,n), n) \parallel \delta))$$
$$\stackrel{5.5.2}{=} \tau_{\{c\}}(\partial_{\{r,s\}}(A(1,n) \parallel \Pi_2(n,n) \parallel \delta))$$
$$\stackrel{SC}{=} \tau_{\{c\}}(\partial_{\{r,s\}}((A(1,n) \parallel \delta) \parallel \Pi_2(n,n)))$$
$$\stackrel{A.6}{=} \tau_{\{c\}}(\partial_{\{r,s\}}(A(1,n) \parallel \Pi_2(n,n)))$$
$$= Sched(n)$$

5. $Sched(n) \stackrel{6.2.4}{=} Sched(1, \varnothing, n) \stackrel{6.2.3}{=} Schedspec(1, \varnothing, n)$.

$\square$

7　Concluding remarks

The experiment can be considered successful: we have brought down Milner's proof to a completely formal level and checked it by computer. Yet we also have to admit that formalising and checking Milner's proof was not a bed of roses.

First, identities that are simple at meta-level are not easy to prove in a formalised setting, e.g. the Π-Expansion lemma. Generally speaking, the identities that were most difficult to prove were those that involve processes which heavily interact with data.

Second, we had to extend μCRL with alphabet axioms and refine our notion of hiding and encapsulation, to be able to formalise the application of the Restriction laws in Milner's proof. It turned out that the formalisation of the Restriction laws (alphabet axioms) in a setting with data was not straightforward. This imposed a considerable delay on our work.

Finally, we had to write out and check a large amount of small proof steps. This is not only hard work, but, again, identities that are trivial at meta-level (and therefore mostly omitted) can sometimes be quite tedious at formal level.

Although the verification was not an easy task, we are confident that by doing more of such protocol verifications we obtain more skill and experience in doing calculations such as given in the paper. Moreover, we believe that proof-checkers can be improved in generating more proof steps by themselves, e.g. by using more advanced *tactics*. This will lead to a situation where proof-checked verification of distributed systems becomes feasible.

A　An overview of the proof theory for μCRL

In [12] a kernel proof system has been given which allows to prove identities about processes with data. This proof system is summarised in A.1. In A.2, we present some basic lemmas which are derived from this system and which we used in the verification of Milner's scheduler. Beside the kernel system we use the so-called *alphabet axioms*. These are presented in appendix B.

14

A.1 The proof system

Table 1 lists the axioms of ACP in μCRL, followed by the axioms for hiding TI, standard concurrency SC and branching bisimulation B. For an explanation of the axioms we refer to [12], except for the following points. We distinguish between *actions* (e.g. $r(i)$ is an action) and *gates*, which are 'incomplete' actions (e.g. r is a gate). The function *label* extracts the gate from an action. The communication axioms, denoted by CF, make use of the function γ. It is defined as follows: $\gamma(a, b) = c$ if $label(a) \mid label(b) = label(c)$ is declared in **comm** and otherwise $\gamma(a, b)$ is undefined.

Table 2 lists the typical μCRL axioms and rules for interaction between data and processes. The axioms for summation are denoted by SUM, the axioms for the conditional by COND and the rules for the booleans by BOOL.

Beside the axioms and rules mentioned above, μCRL incorporates two other important proof principles. First, it supports an principle for induction not only on data but also on data in processes. The second principle is RSP (Recursive Specification Principle) taken from [1] extended to processes with data. Informally, it says that each guarded recursive specification has at most one solution.

A.2 Basic lemmas for μCRL

In this section, we present a number of elementary lemmas (see [9]) which are derived from the proof system given above. These lemmas are used in the verification of the scheduler, but are also interesting in their own right as it is very likely that they are needed in every μCRL verification. The first lemma shows that for applying an induction on a boolean variable b (see appendix C), one only has to check the cases $b = T$ and $b = F$.

Lemma A.1. *(Specialised induction rule for **Bool**).*

$$(p = q)[T/b] \wedge (p = q)[F/b] \rightarrow p = q.$$

Lemma A.2.

1. $x \triangleleft b \triangleright x = x$,

2. $x + x \triangleleft b \triangleright \delta = x$,

3. $x \triangleleft b \triangleright y = x \triangleleft b \triangleright y + x \triangleleft b \triangleright \delta$.

Proof. By Lemma A.1. $\qquad\qquad\square$

The following lemma presents a rule which is derived from the SUM axioms. This rule appears to be a powerful tool to eliminate sum expressions in μCRL calculations.

Lemma A.3. *(Sum Elimination). Let D be a given sort that is equipped with an equality function $eq : D \times D \to \textbf{Bool}$ with the obvious property $eq(d, d) = T$. Then, we have*

$$\Sigma_{d:D}(p \triangleleft eq(d, t) \triangleright \delta) = p[t/d].$$

The next lemma is used for expanding sums in parallel compositions.

Lemma A.4. *(Sum Expansion). If the variable $d : D$ does not occur free in term q, then we have*

1. $\Sigma_{d:D}(a \cdot p \triangleleft c \triangleright \delta) \, \| \, q = \Sigma_{d:D}(a \cdot (p \parallel q) \triangleleft c \triangleright \delta)$.

2. $\Sigma_{d:D}(a(d) \cdot p \triangleleft c \triangleright \delta) \mid b(e) \cdot q = \Sigma_{d:D}((a(d) \mid b(e)) \cdot (p \parallel q) \triangleleft c \triangleright \delta)$

The following proposition is used in Theorem 6.2.1.

15

A1	$x + y = y + x$		CF1	$n_1 \mid n_2 = n_3$ $\qquad$ if $\gamma(n_1, n_2) = n_3$
A2	$x + (y + z) = (x + y) + z$			
A3	$x + x = x$		CF1$'$	$n_1(t_1, \ldots, t_m) \mid n_2(t_1, \ldots, t_m) = n_3(t_1, \ldots, t_m)$
A4	$(x + y) \cdot z = x \cdot z + y \cdot z$			$\qquad$ if $\gamma(n_1, n_2) = n_3$
A5	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$			
A6	$x + \delta = x$		CF2	$a \mid b = \delta$
A7	$\delta \cdot x = \delta$			$\qquad$ if $\gamma(label(a), label(b))$ is undefined
			CF2$'$	$\neg(t_i = t_i') \rightarrow n_1(t_1, \ldots, t_m) \mid n_2(t_1', \ldots, t_m') = \delta$
				$\qquad$ for some $1 \le i \le m$
CM1	$x \parallel y = x \, \underline{\parallel} \, y + y \, \underline{\parallel} \, x + x \mid y$			
CM2	$a \, \underline{\parallel} \, x = a \cdot x$		CF2$''$	$n_1(t_1, \ldots, t_m) \mid n_2(t_1', \ldots, t_{m'}') = \delta$ $\quad$ if $m \ne m'$
CM3	$a \cdot x \, \underline{\parallel} \, y = a \cdot (x \parallel y)$			
CM4	$(x + y) \, \underline{\parallel} \, z = x \, \underline{\parallel} \, z + y \, \underline{\parallel} \, z$			
CM5	$a \cdot x \mid b = (a \mid b) \cdot x$		D1	$\partial_H(a) = a$ $\qquad$ if $label(a) \notin H$
CM6	$a \mid b \cdot x = (a \mid b) \cdot x$		D2	$\partial_H(a) = \delta$ $\qquad$ if $label(a) \in H$
CM7	$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$		D3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
CM8	$(x + y) \mid z = x \mid z + y \mid z$		D4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$
CM9	$x \mid (y + z) = x \mid y + x \mid z$			
TI1	$\tau_I(a) = a$ $\;$ if $label(a) \notin I$		TI3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$
TI2	$\tau_I(a) = \tau$ $\;$ if $label(a) \in I$		TI4	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$
SC1	$(x \, \underline{\parallel} \, y) \, \underline{\parallel} \, z = x \, \underline{\parallel} \, (y \parallel z)$		SC4	$(x \mid y) \mid z = x \mid (y \mid z)$
SC6	$x \parallel \delta = x\delta$		SC5	$x \mid (y \, \underline{\parallel} \, z) = (x \mid y) \, \underline{\parallel} \, z$
SC3	$x \mid y = y \mid x$		SC8	$x \mid (y \mid z) = \delta$
SC7	$x \mid \delta = \delta$			
B1	$x\tau = x$		B2	$z(\tau(x + y) + x) = z(x + y)$

Table 1: ACP-like axioms and rules in μCRL.

SUM1	$\Sigma_{d:D}(p) = p$	if d not free in p
SUM2	$\Sigma_{d:D}(p) = \Sigma_{e:D}(p[e/d])$	if e not free in p
SUM3	$\Sigma_{d:D}(p) = \Sigma_{d:D}(p) + p$	
SUM4	$\Sigma_{d:D}(p_1 + p_2) = \Sigma_{d:D}(p_1) + \Sigma_{d:D}(p_2)$	
SUM5	$\Sigma_{d:D}(p_1 \cdot p_2) = \Sigma_{d:D}(p_1) \cdot p_2$	if d not free in p_2
SUM6	$\Sigma_{d:D}(p_1 \parallel p_2) = \Sigma_{d:D}(p_1) \parallel p_2$	if d not free in p_2
SUM7	$\Sigma_{d:D}(p_1 \mid p_2) = \Sigma_{d:D}(p_1) \mid p_2$	if d not free in p_2
SUM8	$\Sigma_{d:D}(\partial_H(p)) = \partial_H(\Sigma_{d:D}(p))$	
SUM9	$\Sigma_{d:D}(\tau_I(p)) = \tau_I(\Sigma_{d:D}(p))$	

$$\text{SUM11} \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ p_1 = p_2\end{array}}{\Sigma_{d:D}(p_1) = \Sigma_{d:D}(p_2)} \qquad \text{provided } d \text{ not free in the assumptions of } \mathcal{D}$$

COND1	$x \triangleleft T \triangleright y = x$
COND2	$x \triangleleft F \triangleright y = y$

BOOL1	$\neg(T = F)$
BOOL2	$\neg(b = T) \rightarrow b = F$

Table 2: Axioms for summation and conditionals.

Proposition A.5. *Let p be a process.*

$$test(i, X) \rightarrow$$
$$b(i)p + \Sigma_{j:nat}(b(j)p \triangleleft test(j, rem(i, X)) \triangleright \delta) = \Sigma_{j:nat}(b(j)p \triangleleft test(j, X) \triangleright \delta)$$

Proof.

$$\Sigma_{j:nat}(b(j)p \triangleleft test(j, X) \triangleright \delta)$$
$$\overset{\text{C.4.1}}{=} \quad \Sigma_{j:nat}(b(j)p \triangleleft eq(j, i) \text{ or } test(j, rem(i, X)) \triangleright \delta)$$

$$\overset{\text{C.1.2,SUM4}}{=} \quad \Sigma_{j:nat}(b(j)p \triangleleft eq(j, i) \triangleright \delta)$$
$$+ \quad \Sigma_{j:nat}(b(j)p \triangleleft test(j, rem(i, X)) \triangleright \delta)$$

$$\overset{\text{A.3}}{=} \quad b(i)p + \Sigma_{j:nat}(b(j)p \triangleleft test(j, rem(i, X)) \triangleright \delta)$$

$\square$

Proposition A.6.

$$A(i, n) \parallel \delta = A(i, n)$$

Proof.

1. Below we show that the process $\lambda in.A(i,n) \parallel \delta$ is a solution for A. By RSP we then have the desired identity $A(i,n) \parallel \delta = A(i,n)$.

   Note that from the axiom $x \mid \delta = \delta$ (see Table 1) we can derive

   (a) $x \parallel \delta = x \parallel\!\!\!\parallel \delta$

   by using CM1,CM2,A7,A6 (see Table 1).

$$
\begin{aligned}
&A(i,n) \parallel \delta \\
&\overset{\text{(a)}}{=} \quad A(i,n) \parallel\!\!\!\parallel \delta \\
&\overset{\text{Definition}}{=} \quad a(i)s(i)(b(i) \parallel r(i -_n 1))A(i,n) \parallel\!\!\!\parallel \delta \\
&\overset{\text{CM3}}{=} \quad a(i)(s(i)(b(i) \parallel r(i -_n 1))A(i,n) \parallel \delta) \\
&\overset{\text{(a)}}{=} \quad a(i)(s(i)(b(i) \parallel r(i -_n 1))A(i,n) \parallel\!\!\!\parallel \delta) \\
&\overset{\text{CM3}}{=} \quad a(i)s(i)((b(i) \parallel r(i -_n 1))A(i,n) \parallel \delta) \\
&\overset{\text{CM1-2,CF2,A6}}{=} a(i)s(i)((b(i)r(i -_n 1) + r(i -_n 1)b(i))A(i,n) \parallel \delta) \\
&\overset{\text{A4}}{=} \quad a(i)s(i)((b(i)r(i -_n 1)A(i,n) + r(i -_n 1)b(i)A(i,n)) \parallel \delta) \\
&\overset{\text{(a),CM4}}{=} a(i)s(i)(b(i)r(i -_n 1)A(i,n) \parallel\!\!\!\parallel \delta + r(i -_n 1)b(i)A(i,n) \parallel\!\!\!\parallel \delta) \\
&\overset{\text{CM3}}{=} \quad a(i)s(i)(b(i)(r(i -_n 1)A(i,n) \parallel \delta) + r(i -_n 1)(b(i)A(i,n) \parallel \delta)) \\
&\overset{\text{(a),CM3}}{=} a(i)s(i)(b(i)r(i -_n 1)(A(i,n) \parallel \delta) + r(i -_n 1)b(i)(A(i,n) \parallel \delta)) \\
&\overset{\text{A4}}{=} \quad a(i)s(i)(b(i)r(i -_n 1) + r(i -_n 1)b(i))(A(i,n) \parallel \delta) \\
&\overset{\text{CM1-2,CF2,A6}}{=} a(i)s(i)(b(i) \parallel r(i -_n 1))(A(i,n) \parallel \delta)
\end{aligned}
$$

$\square$

Proposition A.7.

$$\tau(x \parallel y) = \tau(\tau x \parallel y)$$

Proof. $\tau(x \parallel y) \overset{\text{CM3}}{=} \tau x \parallel\!\!\!\parallel y \overset{\text{B1}}{=} \tau\tau x \parallel\!\!\!\parallel y \overset{\text{CM3}}{=} \tau(\tau x \parallel y).$ $\square$

B Alphabet axioms

As mentioned in section 6, we sometimes want to encapsulate occurrences of r in a process p which have the natural number i as parameter, while leaving occurrences of $r(i +_n 1)$ unchanged. To this end we extend our proof system with the axioms given in Table 3.

We also add the alphabet axioms given in Table 4 to the system. In this table A, B range over lists of actions and G, G_1, G_2 range over lists of gates. $\varnothing_G$ stands for the empty set of gates and $\varnothing_A$ stands for the empty set of actions. Furthermore:

- $Range(\gamma)$ is the range of the communication function γ. In our case this is $\{c, \hat{a}, \hat{b}\}$.

- $label(A)$ is the set of gates that occur in the list of actions A. For instance $label(< r(i), s(i) >) = \{r, s\}$.

- $Partners(a)$ is the set of (parametrised) actions b that can communicate with a. $Partners'(a)$ is the set of actions b that can communicate with a such that the resulting action c is not equal to a. In our example, we have $Partners(r(i)) = Partners'(r(i)) = \{s(i)\}$.

To our knowledge this is the first time alphabet axioms (Milner calls them Restriction laws) are brought down to a completely formal level such that they can be used for proof checking. For a more general discussion on this subject we refer to [15].

DA1 $\neg(a = b) \rightarrow \partial_{<a>}(b) = b$	TA1 $\neg(a = b) \rightarrow \tau_{<a>}(b) = b$
DA2 $a = b \rightarrow \partial_{<a>}(b) = \delta$	TA2 $a = b \rightarrow \tau_{<a>}(b) = \tau$
DA3 $\partial_{<a>}(x + y) = \partial_{<a>}(x) + \partial_{<a>}(y)$	TA3 $\tau_{<a>}(x + y) = \tau_{<a>}(x) + \tau_{<a>}(y)$
DA4 $\partial_{<a>}(x \cdot y) = \partial_{<a>}(x) \cdot \partial_{<a>}(y)$	TA4 $\tau_{<a>}(x \cdot y) = \tau_{<a>}(x) \cdot \tau_{<a>}(y)$

Table 3: Axioms for hiding and encapsulation actions.

CAA1	$\partial_{Partners'(a)}(x) = x \rightarrow \quad \partial_{<a>}(x \parallel y) = \partial_{<a>}(x \parallel \partial_{<a>}(y))$	
CAA2	$\partial_{Partners(a)}(x) = x \rightarrow \quad \tau_{<a>}(x \parallel y) = \tau_{<a>}(x \parallel \tau_{<a>}(y))$	
CAA3	$\partial_A(\partial_A(x) \parallel \partial_A(y)) = \partial_A(x) \parallel \partial_A(y)$	if $label(A) \cap Range(\gamma) = \varnothing_G$
CAA3′	$\partial_{\varnothing_A}(x) = x$	
CAA3″	$\partial_A(\delta) = \delta$	
CAA3‴	$\partial_A(\tau) = \tau$	
CAA4	$\tau_A(\tau_A(x) \parallel \tau_A(y)) = \tau_A(x) \parallel \tau_A(y)$	if $label(A) \cap Range(\gamma) = \varnothing_G$
CAA4′	$\tau_{\varnothing_A}(x) = x$	
CAA4″	$\tau_A(\delta) = \delta$	
CAA4‴	$\tau_A(\tau) = \tau$	
CAA5	$\partial_{A \cup B}(x) = \partial_A(\partial_B(x))$	
CAA6	$\tau_{A \cup B}(x) = \tau_A(\tau_B(x))$	
CAA7	$\neg(a = b) \rightarrow \tau_{<a>}(\partial_{<b>}(x)) = \partial_{<b>}(\tau_{<a>}(x))$	
CAG7	$\tau_{G_1}(\partial_{G_2}(x)) = \partial_{G_2}(\tau_{G_1}(x))$	if $G_1 \cap G_2 = \varnothing_G$
CAGA1	$\partial_G(x) = \partial_G(\partial_A(x))$	if $label(A) \subseteq G$
CAGA2	$\tau_G(x) = \tau_G(\tau_A(x))$	if $label(A) \subseteq G$

Table 4: The alphabet axioms which we used in the verification of the scheduler.

Using the above mentioned axioms, we shall prove Lemma B.3. This lemma is used in the main theorem and is essential for having Milner's proof formalised in μCRL; it allows to distribute the

encapsulation and hiding operators over the merge operator. (It is an open question whether or not the scheduler can be verified without the use of alphabet axioms in an elegant way, if it is possible at all.) For proving Lemma B.3 we use two auxiliary propositions which are listed below.

Proposition B.1. *Recall that 'Cond' is an abbreviation for '$n \geq 2 \wedge i \geq 1 \wedge i \leq n \wedge X \subseteq fill(1,n) \wedge unique(X)$'.*

1. $\neg eq(j, i +_n 1) \wedge Cond \rightarrow \partial_{<r(i)>}(D(j,n)) = D(j,n)$

2. $\neg test(i +_n 1, X) \wedge Cond \rightarrow \partial_{<r(i)>}(\Pi_D(X,n)) = \Pi_D(X,n)$

3. $\neg test(i +_n 1, X) \wedge Cond \rightarrow \partial_{<r(i)>}(\Pi_E(X,n)) = \Pi_E(X,n)$

4. $test(i +_n 1, X) \wedge Cond \rightarrow$
$$\partial_{<r(i)>}(\Pi_D(in(i,X)^n, n) \parallel \Pi_E(rem(i +_n 1, X), n)) =$$
$$\Pi_D(in(i,X)^n, n) \parallel \Pi_E(rem(i +_n 1, X), n)$$

5. $\neg test(i +_n 1, X) \wedge Cond \rightarrow$
$$\partial_{<r(i)>}(\Pi_D(rem(i +_n 1, in(i,X)^n), n) \parallel \Pi_E(X, n)) =$$
$$\Pi_D(rem(i +_n 1, in(i,X)^n), n) \parallel \Pi_E(X, n)$$

Note that *Cond* can be weakened in some cases.

Proof.

1. Instead of giving the proof, which is a straightforward application of RSP, we remark that this identity is not true if we replace $\partial_{<r(i)>}$ by $\partial_{\{r\}}$. For $\partial_{\{r\}}(D(j,n)) = \partial_{\{r\}}(r(j -_n 1)A(j,n)) = \partial_{\{r\}}(A(j,n)) \neq D(j,n)$.

2. With induction on X.

   - $X = \varnothing$:
   $$\partial_{<r(i)>}(\Pi_D(\varnothing, n)) = \partial_{<r(i)>}(\delta) \stackrel{\mathrm{CAA3''}}{=} \delta = \Pi_D(\varnothing, n)$$

   - $X = in(j, Y)$:
   $$\partial_{<r(i)>}(\Pi_D(in(j,Y), n))$$

   $$\stackrel{5.1.1}{=} \quad \partial_{<r(i)>}(D(j,n) \parallel \Pi_D(Y, n))$$

   $$= \quad \partial_{<r(i)>}(D(j,n) \parallel \partial_{<r(i)>}(\Pi_D(Y, n)))$$

   by I.H. and the fact that $\neg test(i +_n 1, in(j,Y)) \wedge Cond \rightarrow \neg test(i +_n 1, Y)$

   $$= \quad \partial_{<r(i)>}(\partial_{<r(i)>}(D(j,n)) \parallel \partial_{<r(i)>}(\Pi_D(Y, n)))$$

   by B.1.1 and the fact that $\neg test(i +_n 1, in(j,Y)) \wedge Cond \rightarrow \neg eq(j, i +_n 1)$

   $$\stackrel{\mathrm{CAA3}}{=} \quad \partial_{<r(i)>}(D(j,n)) \parallel \partial_{<r(i)>}(\Pi_D(Y, n))$$

   $$= \quad D(j,n) \parallel \Pi_D(Y, n)$$

   by applying the second and third step in reverse direction

$$\overset{5.1.1}{=} \qquad \Pi_D(in(j,Y),n)$$

3. Analogous to (2). Note that B.1.1 also holds for D replaced by E.

4. Similar to (2).

5. Similar to (2).

$\square$

Proposition B.2.

1. $\neg eq(j,i) \rightarrow \ \partial_{<s(i)>}(D(j,n)) = D(j,n)$

2. $\neg test(i,X) \wedge Cond \rightarrow \ \partial_{<s(i)>}(\Pi_D(X,n)) = \Pi_D(X,n)$

3. $\neg test(i,X) \wedge Cond \rightarrow \ \partial_{<s(i)>}(\Pi_E(X,n)) = \Pi_E(X,n)$

4. $\neg test(i,X) \wedge Cond \rightarrow$
$$\partial_{<s(i)>}(\Pi_D(in(i,X)^n,n) \parallel \Pi_E(rem(i+_n 1,X),n)) =$$
$$\Pi_D(in(i,X)^n,n) \parallel \Pi_E(rem(i+_n 1,X),n)$$

5. $\neg test(i,X) \wedge Cond \rightarrow$
$$\partial_{<s(i)>}(\Pi_D(rem(i+_n 1,in(i,X)^n),n) \parallel \Pi_E(X,n)) =$$
$$\Pi_D(rem(i+_n 1,in(i,X)^n),n) \parallel \Pi_E(X,n)$$

Proof. Analogous to the proof of B.1. $\square$

Lemma B.3. *Let '$\mathcal{F}(p)$' be notation for '$\tau_{<c(i)>}(\partial_{<s(i),r(i)>}(p))$', where p is an arbitrary μCRL process.*

1. $test(i+_n 1,X) \wedge \neg test(i,X) \wedge Cond \rightarrow$
$$\tau_{\{c\}}(\partial_{\{r,s\}}(p \parallel (\Pi_D(in(i,X)^n,n) \parallel \Pi_E(rem(i+_n 1,X),n)))) =$$
$$\tau_{\{c\}}(\partial_{\{r,s\}}(\mathcal{F}(p) \parallel (\Pi_D(in(i,X)^n,n) \parallel \Pi_E(rem(i+_n 1,X),n))))$$

2. $\neg test(i+_n 1,X) \wedge \neg test(i,X) \wedge Cond \rightarrow$
$$\tau_{\{c\}}(\partial_{\{r,s\}}(p \parallel (\Pi_D(rem(i+_n 1,in(i,X)^n),n) \parallel \Pi_E(X,n)))) =$$
$$\tau_{\{c\}}(\partial_{\{r,s\}}(\mathcal{F}(p) \parallel (\Pi_D(rem(i+_n 1,in(i,X)^n),n) \parallel \Pi_E(X,n))))$$

Proof.

1. Write 'Q' for '$\Pi_D(in(i,X)^n,n) \parallel \Pi_E(rem(i+_n 1,X),n)$.

$$\tau_{\{c\}}(\partial_{\{r,s\}}(p \parallel Q))$$

$$\overset{\mathrm{CAGA1}}{=} \qquad \tau_{\{c\}}(\partial_{\{r,s\}}(\partial_{<r(i),s(i)>}(p \parallel Q)))$$

$$\overset{\mathrm{CAA1}}{=} \qquad \tau_{\{c\}}(\partial_{\{r,s\}}(\partial_{<r(i),s(i)>}(\partial_{<s(i)>}(p) \parallel Q)))$$

using Proposition B.1.4, see note I below

$$\stackrel{\mathrm{CAA1}}{=} \quad \tau_{\{c\}}(\partial_{\{r,s\}}(\partial_{<r(i),s(i)>}(\partial_{<r(i),s(i)>}(p) \parallel Q)))$$

using Proposition B.2.4, see note II below

$$\stackrel{\mathrm{CAGA1}}{=} \quad \tau_{\{c\}}(\partial_{\{r,s\}}(\partial_{<r(i),s(i)>}(p) \parallel Q))$$

$$\stackrel{\mathrm{CAG7}}{=} \quad \partial_{\{r,s\}}(\tau_{\{c\}}(\partial_{<r(i),s(i)>}(p) \parallel Q))$$

$$\stackrel{\mathrm{CAGA2}}{=} \quad \partial_{\{r,s\}}(\tau_{\{c\}}(\tau_{<c(i)>}(\partial_{<r(i),s(i)>}(p) \parallel Q)))$$

$$\stackrel{\mathrm{CAA2}}{=} \quad \partial_{\{r,s\}}(\tau_{\{c\}}(\tau_{<c(i)>}(\mathcal{F}(p) \parallel Q)))$$

using CAA3', see note III below

$$\stackrel{\mathrm{CAGA2}}{=} \quad \partial_{\{r,s\}}(\tau_{\{c\}}(\mathcal{F}(p) \parallel Q))$$

$$\stackrel{\mathrm{CAG7}}{=} \quad \tau_{\{c\}}(\partial_{\{r,s\}}(\mathcal{F}(p) \parallel Q))$$

<u>note I:</u> $\partial_{Partners'(s(i))}(Q) = \partial_{<r(i)>}(Q) \stackrel{\mathrm{B.1.4}}{=} Q$.

<u>note II:</u> $\partial_{Partners'(r(i))}(Q) = \partial_{<s(i)>}(Q) \stackrel{\mathrm{B.2.4}}{=} Q$.

<u>note III:</u> $\partial_{Partners(c(i))}(Q) = \partial_{\varnothing_A}(Q) \stackrel{\mathrm{CAA3'}}{=} Q$.

2. Similar to (1) by using B.1.5, B.2.5 instead of B.1.4, B.2.4.

$\square$

C   Elementary data types

Below, we present the data identities we needed in the scheduler verification. Although all these results have been proof-checked we do not present the proofs here, since they are standard.

C.1   About booleans

sort	**Bool**
func	$T, F :\rightarrow$ **Bool**

func	not : **Bool** $\rightarrow$ **Bool**
	and : **Bool** $\times$ **Bool** $\rightarrow$ **Bool**
	or : **Bool** $\times$ **Bool** $\rightarrow$ **Bool**
var	$b, b_1, b_2, b_3 :$ **Bool**
rew	$\mathrm{not}(T) = F$
	$\mathrm{not}(F) = T$
	T and $b = b$
	F and $b = F$
	T or $b = T$
	F or $b = b$

Lemma C.1.

1. $x \triangleleft b \triangleright y = y \triangleleft \mathrm{not}(b) \triangleright x$,

2. $x \triangleleft b_1 \text{ or } b_2 \triangleright \delta = x \triangleleft b_1 \triangleright \delta + x \triangleleft b_2 \triangleright \delta$,

C.2 About natural numbers

$$
\begin{array}{ll}
\textbf{sort} & nat \\
\textbf{func} & 0 :\to nat \\
& S, P : nat \to nat \\
& +, -, : nat \times nat \to nat \\
& eq, \geq, \leq, <, >: nat \times nat \to \textbf{Bool} \\
& if : \textbf{Bool} \times nat \times nat \to nat \\
\textbf{var} & n, m, z : nat \\
\textbf{rew} & P(0) = 0 \\
& P(S(n)) = n \\
& n + 0 = n \\
& n + S(m) = S(n + m) \\
& n - 0 = n \\
& n - S(m) = P(n - m) \\
& eq(0, 0) = T \\
& eq(0, S(n)) = F \\
& eq(S(n), 0) = F \\
& eq(S(n), S(m)) = eq(n, m) \\
& n \geq 0 = T \\
& 0 \geq S(n) = F \\
& S(n) \geq S(m) = n \geq m \\
& n \leq m = m \geq n \\
& n > m = n \geq S(m) \\
& n < m = S(n) \leq m \\
& if(T, n, m) = n \\
& if(F, n, m) = m
\end{array}
$$

We write $n \leq m$ for $n \leq m = T$. Idem for $\geq, >$ and $<$. We write $eq(n, m)$ for $eq(n, m) = T$. We write 1 for $S(0)$ and 2 for $S(S(0))$. We write $i - 1$ for $P(i)$ and $i - 2$ for $P(P(i))$.

Lemma C.2.

$$eq(n, m) = T \leftrightarrow n = m$$

C.3 About modulo arithmetic

The following definition is due to Willem Jan Fokkink.

$$
\begin{array}{ll}
\textbf{func} & \mathrm{mod} : nat \times nat \to nat \\
& + : nat \times nat \times nat \to nat \\
\textbf{var} & i, j, n : nat \\
\textbf{rew} & i \bmod 0 = i \\
& i \bmod n = if(eq(i, 0), n, if(i > n, (i - n) \bmod n, i)) \\
& i +_n j = (i + j) \bmod n \\
& i -_n j = (i - j) \bmod n
\end{array}
$$

Note that we defined a slightly non-standard modulo function to follow Milner's proof as close as possible. In particular, we need our functions to have values in the positive natural numbers. The usual definition of the modulo function yields e.g. 2 mod 2 = 0; our (and Milner's) definition yields 2 mod 2 = 2.

Lemma C.3.

1. $i \bmod 1 = 1$

2. $n \geq 2 \wedge 1 \leq i \wedge i \leq n \rightarrow (i +_n 1) -_n 1 = i$

3. $n \geq 2 \wedge 1 \leq i \wedge i \leq n \rightarrow (i -_n 1) +_n 1 = i$

4. $n \geq 2 \wedge i \leq n \rightarrow \neg eq(i, i +_n 1)$ *(used in C.4.11)*

C.4 About lists of naturals

$$
\begin{array}{ll}
\textbf{sort} & list \\
\textbf{func} & \varnothing :\rightarrow list \\
& in, rem, \ ^n : nat \times list \rightarrow list \\
& test : nat \times list \rightarrow \textbf{Bool} \\
& hd : list \rightarrow nat \\
& tl : list \rightarrow list \\
& if : \textbf{Bool} \times list \times list \rightarrow list \\
& empty, unique : list \rightarrow \textbf{Bool} \\
& fill : nat \times nat \rightarrow list \\
& - : list \times list \rightarrow list \\
& \subseteq, perm : list \times list \rightarrow \textbf{Bool} \\
\textbf{var} & i, j, k, n, m : nat \\
& X, Y : list \\
\textbf{rew} & test(j, \varnothing) = F \\
& test(j, in(k, X)) = if(eq(j, k), T, test(j, X)) \\
& rem(j, \varnothing) = \varnothing \\
& rem(j, in(k, X)) = if(eq(j, k), X, in(k, rem(j, X))) \\
& hd(\varnothing) = 0 \\
& hd(in(j, X)) = j \\
& tl(\varnothing) = \varnothing \\
& tl(in(j, X)) = X \\
& empty(\varnothing) = T \\
& empty(in(j, X)) = F \\
& fill(m, n) = if(n < m, \varnothing, if(eq(n, 0), in(0, \varnothing), in(n, fill(m, P(n))))) \\
& X^n = fill(1, n) - X \\
& X - \varnothing = X \\
& X - in(j, Y) = rem(j, X - Y) \\
& \varnothing \subseteq X = T \\
& in(j, X) \subseteq Y = test(j, Y) \text{ and } X \subseteq Y \\
& unique(\varnothing) = T \\
& unique(in(j, X)) = if(test(j, X), F, unique(X)) \\
& perm(X, Y) = X \subseteq Y \text{ and } Y \subseteq X
\end{array}
$$

Lemma C.4.

1. $test(i, X) \rightarrow (test(j, X) = eq(i, j) \text{ or } test(j, rem(i, X)))$,

2. $in(1, \varnothing)^n = fill(2, n)$,

3. $\neg eq(i, j) \to in(i, rem(j, Y)) = rem(j, in(i, Y))$,

4. $eq(i, j) \to rem(i, in(j, Y)) = Y$,

5. $n \geq 1 \to fill(2, S(n)) = in(S(n), fill(2, n))$,

6. $rem(hd(X), X) = tl(X)$,

7. $test(hd(X), X) = \text{not}(empty(X))$.

8. $(test(i, X) \wedge X = in(j, Y) \wedge \neg eq(i, j)) \to test(i, Y)$.

9. $rem(i, rem(j, X)) = rem(j, rem(i, X))$,

10. $in(i +_n 1, in(i, X))^n = rem(i +_n 1, in(i, X)^n)$.

11. $n \geq 2 \wedge i \leq n \to rem(i +_n 1, in(i, X)) = in(i, rem(i +_n 1, X))$

12. $test(i, X) \to in(i, rem(i, X))^n = X^n$.

D About Coq

Coq is a proof assistant based on the *formulas as types, proofs as terms* paradigm (see [7]). In this paradigm, a formula is translated into a type in a typed lambda calculus and proofs of this formula are translated into lambda-terms of the corresponding type. Coq is an assistant in the sense that the proof is built up step by step by the user, while the computer checks the correctness of each step. Small proof steps can be done automatically by Coq. The actual construction of the lambda-term (the proof) is hidden from the user: the user just enters commands which are close to expressions in traditional proofs. Therefore the reasoning in Coq is very similar to reasoning in 'ordinary' mathematics.

The type theory underlying Coq is an extension of the Calculus of Constructions (see [3]) with Inductive Types (see [18]). The translation of a data type (with a set of constructors) into an inductive type (when this is possible) has the effect that the constructors of the inductive type are independent and an induction principle and a recursion scheme for this inductive type are generated.

As an illustration of the first feature, after the declaration of the inductive type bool with two constructors true and false, the inequality of true and false holds by definition, whereas in μCRL it is given by the axiom BOOL1. The latter two features enable the user to reason with induction over the data type and to define functions by recursion over the data type.

We have tried to use these facilities as much as possible. For instance, the sort *list* in μCRL is implemented as an inductive type *List* in with two constructors *nil* (for the empty list) and *in* (for appending an element to a list) in Coq. The functions on lists are not defined equationally like in μCRL but directly using the recursion scheme for *List*. In some cases this allows for simpler proofs. E.g. in μCRL, the identity $test(2, in(1, in(1, (in(2, \varnothing))))) = T$ is proved in more than three steps. In Coq this is done in one step since it is internally computed that $test(2, in(1, in(1, (in(2, \varnothing)))))$ equals T.

As an example of the proof checking we now give the development which corresponds to the proof of Proposition A.5. A proof development consists of a series of commands (or 'tactics') entered by the user, which (internally) generate the λ-term that corresponds to the proof in the paper. The command Goal *name* indicates that we are going to prove the 'goal' *name*. Each subsequent tactic is applied to the current goal and generates some (possibly none) new subgoals.

```
Goal (i:nat)(X:Queue)(p:nat->proc)
  (<bool>(Testn i X)=true)-><proc>
    (sum nat [j:nat]
      (cond
        (seq (ia nat b j) (p j))
```

```
          (Testn j X)
          Delta))=
    (alt
      (seq (ia nat b i) (p i))
      (sum nat [j:nat]
        (cond
          (seq (ia nat b j) (p j))
          (Testn j (Rem i X))
          Delta))).
Intros i X p H.
Rewrite <- (A_3_for_nat i [d:nat](seq (ia nat b d) (p d))).
Rewrite <- SUM4.
Apply SUM11.
Intro d.
Rewrite <- C_1_2.
Rewrite (C_4_1 i d X).
Rewrite sym_Eqn.
Auto.
Assumption.
Save PnA_5.
```

We briefly explain the notation. The expression (x:A)P is notation for $\forall x{:}A.P$ ('for all x in A, P is true'). The term A->B denotes 'if A then B', if A and B are taken as propositions, or the type of functions from A to B, if A and B are taken as sets. For instance, nat->proc is the type of processes that have a natural number as parameter. Next, (M N) denotes the application of M to N and [x:A]M denotes $\lambda x{:}A.M$.

Furthermore, <A>a=b denotes that a and b are of type A and equal with respect to the equality for that type, (sum [d:D]p) denotes $\sum_{d:D} p$, cond denotes the ($\_\triangleleft\_\triangleright\_$)-operator, seq denotes $\cdot$ (sequential composition), (ia A b j) denotes the process which consists of action b with datum j of type A (ia is just a constant of the right type). Finally, Testn denotes *test*, Delta denotes δ, alt denotes $+$ (choice), Rem denotes *rem* and Eqn denotes the *eq* function on natural numbers.

We now explain the tactics. The command Intros i X p H has the effect that i, X and p are locally declared variables ('let i, X and p be given') and that we have assumed that the boolean (Testn i X) equals true. Recall that proofs are represented by terms and in this setting 'proofs' of assumptions are just free variables of the corresponding type. So H is a free variable of type <bool>(Testn i X)=true. We say that 'i, X, p, and H are added to the context'.

A command of the form Rewrite *name* has the effect that if *name* denotes a proof of <A>a=b then occurrences of a in the current goal are replaced by b. The command Rewrite <- *name* has the opposite effect. The names given in this example refer to the corresponding result in the paper, except for A_3_for_nat, which refers to Lemma A.3 (Sum Elimination) specialised to the data type *nat*. The name sym_Eqn refers to the proof of the symmetry of equality on natural numbers. Note that in the proof in the paper the application of this fact (in the first step) is left implicit.

After the command Rewrite sym_Eqn the goal is simply the identity of two expressions which are literally equal. This goal is solved automatically by Auto. The proof ends with an Assumption command. This is because the application of Lemma C.4.1 requires that $test(i, X) = T$, a condition which is fulfilled since we are working under this assumption. So Coq inspects the context to find the required assumption H. Thus it answers with Goal proved!, after which the result is stored by the command Save PnA_5.

References

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

[2] 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.

[3] T. Coquand and G. Huet. The calculus of constructions. *Information and Control*, 76:95–120, 1988.

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

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

[6] J.-C. Fernandez, A. Kerbrat and L. Mounier. Symbolic Equivalence Checking. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference*, CAV '93, Elounda, Greece, volume 697 of *Lecture Notes in Computer Science*, pages 85–97. Springer-Verlag, 1993.

[7] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1989.

[8] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989.

[9] J.F. Groote and H. Korver. A correctness proof of the bakery protocol in μCRL. Technical Report Logic Group Preprint Series No. 80, Utrecht University, 1992.

[10] J.F. Groote and J.C. van de Pol. A bounded retransmission protocol for large data packets. A case study in computer checked verification. Technical Report 100, Logic Group Preprint Series, Utrecht University, October 1993.

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

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

[13] J.F. Groote and A. Ponse. μCRL: A base for analysing processes with data. In E. Best and G. Rozenberg, editors, *Proceedings 3rd Workshop on Concurrency and Compositionality, Goslar, GMD-Studien Nr. 191*, pages 125–130. Universität Hildesheim, 1991.

[14] L. Helmink, M.P.A. Sellink, and F. Vaandrager. Proof-checking a data link protocol. 1993. To appear.

[15] H. Korver. Mechanised Alphabet Reasoning in Process Algebra. This thesis.

[16] H. Korver and J. Springintveld. A Computer-Checked Verification of Milner's Scheduler. Technical Report Logic Group Preprint Series No. 101, Utrecht University, November, 1993.

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

[18] C. Paulin-Mohring. Inductive definitions in the system Coq. Rules and properties. In M. Bezem and J.F. Groote, editors, *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications, TLCA '93,* Utrecht, The Netherlands, volume 664 of *Lecture Notes in Computer Science,* pages 328–345. Springer-Verlag, 1993.

[19] M.P.A. Sellink. Verifying process algebra proofs in type theory. Technical Report Logic Group Preprint Series No. 87, Utrecht University, 1993.

[20] N.V. Stenning. A data transfer protocol. *Computer Networks.* 1:99–110, 1976.

[21] A.S. Tanenbaum. *Computer networks.* Prentice-Hall International, Englewood Cliffs, 1989.