A Case Study for the ERIL Project

Carron Kirkwood*
Department of Computing Science
University of Glasgow
Glasgow, G12 8QQ.
email: carron@dcs.glasgow.ac.uk

August 13, 1997

Abstract

We examine an abstraction of a real communications problem in order to identify some of the problems involved in satisfying the verification requirements of such a system. The example is described using LOTOS[ISO88], and the RRL term rewriting system[KZ87] is used to carry out the proofs.

1 Introduction

A common problem in the verification of real systems is: given a specification and an implementation, prove that some sort of relationship holds between the two. Often the problem is further complicated by the implementation having been written with little or no reference to the specification, rather than having been formally derived from it. The main aim of our work on this case study is to discover, and hopefully solve, some of the problems encountered in this process. The automation of the proofs required was a further aim.

The example presented in the case study is an abstraction of a real communications problem involving three communicating processes at OSI Network level. The example is small, allowing the exploration of the problems of verifying the system without being overwhelmed by size or detail. The original example is due to Jeremy Dick[Dic90].

The example is presented in section 2. An informal overview of the whole system is given, followed by separate sections giving informal and formal descriptions of the specification and implementation of the system. The formal descriptions are written in LOTOS[ISO88]. Section 3 is concerned with the verification requirements: what they mean, and how they can be proved to be satisfied. The process of automating these proofs, and the system used (the term rewriting system RRL[KZ87]), is described in section 4; the proofs themselves are discussed in section 5. These first attempts at proving that the verification requirements are satisfied fail because the gap between specification and implementation is too great. This problem is rectified and the final proofs are discussed in section 6. Possible extensions to the case study are discussed in section 7 and a review of the tools used is given in section 8. Our conclusions and ideas for further work are given in section 9. There are two appendices: appendix A gives the proofs omitted from the main text, and appendix B gives the laws from [ISO88] which have been used in the RRL rule sets.

*Funded by SERC grant gr/f 35371/ 4/1/1477, Verification Techniques for LOTOS.

2 The Example

2.1 Informal Overview of the System

There are four communicating processes: A, B, C and D, shown in figure 1. In the diagram, a box represents a process, and a ○→ represents a message (being sent in the direction of the arrow).

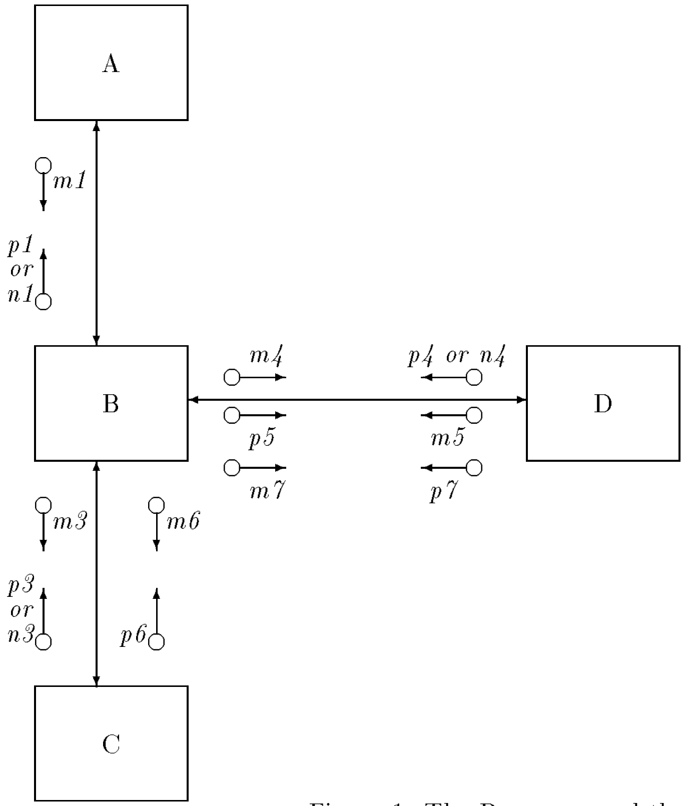

Figure 1: The Processes and their messages

Process A requests a service from B; in order to satisfy that service, B must request a service from process C, then communicate with process D. B has an internal timer which "times out" if D does not reply to its communication within a previously set time limit. B must send deallocation messages to C and D when they are no longer required. The messages may be interpreted as follows:

m1 — A requests a service from B
m3 — B requests a service from C
m4 — B requests communication with D
m5 — D sends a message to B
m6 — B deallocates C
m7 — B deallocates D

Positive and negative acknowledgments to these messages are also sent.[1]

To help illuminate the system, the following possible interpretation is provided: A is a user wishing to log-on to a system with a username and a password. C takes a username and checks that it is valid. D takes a valid username, acknowledges receipt of the name, and then returns the

[1]Note that some messages only require a positive acknowledgment, while others require both positive and negative acknowledgments (see figure 1) — this is to do with the nature of the messages which they acknowledge, e.g. it does not make any sense to allow C to respond in a negative way to the message *m6* "Service terminated".

corresponding password. B co-ordinates these activities to ascertain if A is a valid user and has supplied the correct password.

Two formal descriptions of the system are given below: a group of protocols which make up the specification, and a group of processes which make up the implementation. Note that inconsistencies may be found between the way the specification describes something and the way the implementation describes the same thing. This is because the implementation was not formally derived from the specification and one of the problems considered in this document is that of trying to reconcile any differences between the two.

2.2 Protocols

Communication in the system is governed by protocols $P1$, $P2$ and $P3$. Each protocol describes the interface between just two of the processes in the system, e.g. $P1$ describes the interface between A and B, ignoring C and D.

$P1$: A sends *m1* to B, which must be acknowledged by *p1* or *n1*.

$P2$: B sends *m3* to C which must be acknowledged by *p3* or *n3*. Following *p3*, B may or may not send *m6* to C which must be acknowledged by *p6*.

$P3$: B sends *m4* to D which must be acknowledged by *p4* or *n4*. After *p4*, D may or may not send *m5* to B. *m5* must be acknowledged by *p5*. Also after *p4*, B may or may not send *m7* to D. *m7* is acknowledged by *p7*.

The protocols are described formally in Basic LOTOS, i.e. the process algebra part of LOTOS. No data types are required. The reader is assumed to be familiar with LOTOS or a related process algebra such as CCS[Mil89] or CSP[Hoa85]. The language constructs used for the descriptions are: **exit** denoting successful termination, **;** denoting action sequencing, and [] denoting choice between two process expressions. Note that in the remainder of this document, the term *processes* will be used to refer to the implementation part of the case study example. LOTOS processes will be referred to as such, or as process expressions.

In these descriptions we assume the carrier is faithful and that no messages or acknowledgements are lost or corrupted.

```
process P1 := m1; ( n1; exit
                    [] p1; exit)
               endproc

process P2 := m3; ( n3; exit
                    [] p3; (exit [] m6; p6; exit))
               endproc

process P3 :=   exit
               [] m4; (  n4; exit
                      [] p4; (  exit
                             [] m7; p7; (exit [] m5; p5; exit)
                             [] m5; p5; (exit [] m7; p7; exit)))
                endproc
```

Note that the protocols, and later the processes, could/should be described recursively; this is ignored at the moment, the simpler finite case being dealt with first. Having finite LOTOS processes instead of recursive ones results in the "skip" branch given in $P3$; this expresses the notion that $P3$ is not always activated. Also notice that the protocols do not always describe "sensible" traces, given our intuitive understanding of the system, e.g. it makes no sense for the message *m5* to be sent after the deallocation message *m7* is sent.

2.3 Processes

The implementation of the system is achieved by four interacting processes. B is described last since it is more complex than, and relies on, the others.

process A: A sends *m1* to B. After this message B sends either *p1* or *n1* to A, indicating success or failure of the transaction respectively.

process C: C receives *m3* from B to which it replies either *p3* or *n3*. If *p3* is sent then C expects an *m6* deallocation message, to which it replies *p6*.

process D: D receives *m4* from B, to which it replies *p4*, and the transaction continues, or *n4*, and the transaction terminates. After *p4*, D sends *m5* to B, expecting *p5* in response, then deallocation by *m7*, to which D replies *p7*. The transaction may be terminated if D receives *m7* before it sends *m5*, i.e. the timer has expired causing B to terminate the transaction.

process B: In a successful execution B receives *m1* from A, allocates C with *m3 p3* and D with *m4 p4*, then sets a timer as D must send *m5* within some time limit. When *m5* arrives the timer is cancelled and B replies with *p5*. C and D are deallocated by *m6 p6* and *m7 p7* respectively. Finally B signals the success of the transaction by sending *p1* to A.

This sequence of actions may fail in a number of ways. Either C or D could refuse to participate by returning negative acknowledgments (*n3* or *n4*), or D might not send *m5* within the time period, in which case the timer "times out". In these cases B replies *n1* to A. Deallocation of C and D occurs if and only if they originally agreed to participate in the transaction.

LOTOS is again used for the formal descriptions. A, C, and D are all as simple as possible (they are just cut down versions of the protocols $P1$, $P2$ and $P3$). The specification of the co-ordinating process B is more interesting.

```
process A := m1; ( n1; exit
                 [] p1; exit)
             endproc

process C := m3; ( n3; exit
                 [] p3; m6; p6; exit)
             endproc

process D :=  exit
             [] m4; ( n4; exit
                    [] p4; (  m5; p5; m7; p7; exit
                           [] m7; p7; exit))
             endproc

process B :=
  m1; m3; ( n3; n1; exit
          [] p3; m4; ( n4; m6; p6; n1; exit
                     [] p4; set; ( timeout; m6; p6; m7; p7; n1; exit
                                 [] m5; tcancel; p5; m6; p6; m7; p7; p1; exit)))
  endproc
```

This concludes the description of the example. The processes as described above, and the protocols as described in the previous section, will be used in the rest of the document. Possible extensions to the example are discussed in section 7. We now look at what sort of properties we might wish to prove about this system.

3 The Verification Requirements

For this case study there are two verification requirements:

1. Prove that the specification of the system (the protocols $P1$, $P2$ and $P3$) is satisfied by the implementation (the processes A, B, C and D).

2. Prove that the resources are deallocated correctly.

The terms used above are deliberately vague since the aim is to explore different possible interpretations of the verification requirements. These are now examined in more detail.

3.1 The Implementation Satisfies the Specification

The problem here is how to define the terms "specification", "implementation" and "satisfies". The first two ought to be straightforward since the protocols and the processes have been given, but these are only the bones of the description. For example, the protocols form the specification, but how they should be combined, or indeed if they should be combined, is not mentioned. The same is true of the processes and the implementation.

Suppose the protocols are to be combined to form the specification and the processes combined to form the implementation. This verification requirement then becomes:

$$\begin{array}{c}\text{A combined with B combined with C combined with D} \\ \text{satisfies} \\ P1 \text{ combined with } P2 \text{ combined with } P3\end{array} \tag{1}$$

where "combined with", which may be different for processes as opposed to protocols, has yet to be defined.

Another alternative exploits the modular way in which the protocols and the processes have been defined by examining each facet of the interaction separately.

$$\text{A combined with B satisfies } P1 \tag{2}$$
$$\text{C combined with B satisfies } P2 \tag{3}$$
$$\text{D combined with B satisfies } P3 \tag{4}$$

Since each equation above yields a boolean, the results can be combined using a boolean operator. If & is used the problem then becomes

"is it correct to say ((2) & (3) & (4)) $\Rightarrow$ (1)"

In this case, since $P1, P2$ and $P3$ are all concerned with distinct facets of the communication of the system, it seems likely that the verification can safely be split into parts. Note that this really depends on choosing the right methods of splitting up the system, hiding unimportant events, making individual proofs, and recombining results.

Further Defining our Terms The definition of "specification" and "implementation" given above rely on the definition of "combined with". This can be viewed as dependent on the formalism chosen to describe the system; our choice is given later, in section 4.1. To a certain extent the same is true of "satisfies". However, this definition can also be discussed more abstractly.

To say that the implementation satisfies its specification some sort of relation between the two must be displayed. The particular sort of relation (e.g. equivalence, refinement etc.) will depend on the sort of decisions made in the step between the specification and the implementation. For example, the implementation may resolve some choices left open in the specification, it may add some information about how to perform a particular task, or it may substitute one method of performing a task with a (probably more efficient) method of performing that task. When a larger step is made between the specification and its implementation we may expect all of these sorts of decision.

To investigate the difference between the specification and its implementation their observable behaviour must be examined, i.e. the events they use, and the way in which those events are ordered. In steps which involve adding information the observable behaviour may change. In these cases we assume some mechanism for ignoring these particular events can be used.

In terms of events, two things about the relationship between the specification and the implementation must be considered:

1. what the implementation does is allowed according to the specification,

2. what the implementation refuses to do can be refused by the specification.

The first condition ensures that the implementation does not perform events which cannot be performed by the specification, and the second condition ensures that the implementation does not deadlock in situations in which the specification would not deadlock. These criteria and various notions of what "implementation" means are discussed in [BSS87].

In some cases the relation used will be an equivalence or a congruence relation, i.e. where the processes must implement *all* the alternatives set out in the protocols, while in other cases a refinement relation will suffice. The relations which can be used in conjunction with LOTOS are discussed in section 4.1.

3.2 Deallocation of Resources

The second verification requirement is that all resources be correctly deallocated. The resources in this system are the processes C and D. If either of these are allocated at some point in the execution, i.e. a *p3 or p4* message is sent, then they must subsequently be deallocated, i.e. the message sequences *m6 p6* or *m7 p7* are sent. This leaves the resources free to be used by other entities.

There are several ways if approaching this requirement:

- A count could be kept of the number of allocation events; this must be equal to the number of deallocation pairs. This could be achieved by simulation of the process.

- The deallocation sequence could be specified formally. We then have to try to prove that the processes are an implementation of the new specification. The specification could be written using LOTOS, or perhaps another formalism, such as temporal logic.

The remainder of the document concentrates on satisfying the first verification requirement. The second is solved as a side-effect of the eventual solution to this problem.

4 Making the Example Concrete

One of the aims of the case study exercise was to automate any proofs required. This section gives a brief overview of the mechanics of that process.

4.1 Writing the Verification Requirements in LOTOS

Three decisions must be made in transforming the verification requirement given in section 3.1 into LOTOS, namely, how to interpret "combined with", the hiding of events, and "satisfies".

- "combined with" is defined by the general parallelism operator of LOTOS. This operator takes two process expressions and a list of events. The list specifies the events on which the process expressions must synchronise. The syntax for this operator is $P|[eventlist]|Q$, where P and Q are process expressions.

- The **hide** operator is used to restrict the processes to protocol events only. This operator takes a process expression and a list of events to be hidden. Hidden events are treated like the internal event **i**; they are unobservable and occur instantaneously. The syntax is **hide** *eventlist* **in** P.

- The interpretation of "satisfies" requires more thought (and perhaps more experimentation) as LOTOS has a wide range of equivalences. Starting with the strongest (i.e. makes the least identifications) these are:

  Strong Bisimulation Equivalence This relation requires events to be matched exactly. All events are treated in this way, including the internal event. Given the use of the **hide** operator which converts hidden events into instances of the internal event, an equivalence which ignores these events is required.

  Weak Bisimulation Equivalence This relation requires all events except the internal event to be matched exactly. The internal event is given its status as a special, unobservable, event and can be matched by zero or more internal events. This relation does not preserve the substitution property in all LOTOS contexts, i.e. two process expressions may have the same observable behaviour, but their internal structure could cause them to behave differently when in combination with other process expressions.

  Weak Bisimulation Congruence This is the largest congruent relation contained in weak bisimulation equivalence. Most, but not necessarily all, of the internal events created by the use of **hide** can be removed by weak bisimulation congruence laws.

  Testing Equivalence This relation is generated by a refinement relation over process expressions called **red**. B_1 **red** B_2 says that B_1 is a deterministic reduction of B_2, which may be interpreted as B_1 "implements" B_2. The equivalence generated by **red** is not a congruence relation.

  Testing Congruence This relation is generated by a second refinement relation over process expressions called **cred**, which is the largest sub-relation of **red** with the property of congruence. This means that **cred** has the substitution property with respect to LOTOS contexts.

  Trace Equivalence This says that two process expressions are equivalent if their trace sets, i.e. their sequences of actions, are the same. This corresponds to saying that two finite state automata are equivalent if their output languages are the same. This does not give a satisfactory interpretation of "satisfies" since deadlock properties are not preserved, i.e. two process expressions may be trace equivalent but one may deadlock after a trace s while the other does not.

  In summary, trace equivalence is rejected because it makes too many identifications, strong bisimulation equivalence because it does not make enough. We also reject weak bisimulation equivalence and testing equivalence since they do not guarantee that two process expressions which are identified will behave in the same way in all contexts. Since our system will probably have to interact with other systems this is not acceptable, therefore the most likely interpretation of "satisfies" is weak bisimulation congruence or testing congruence.

The next section describes the software used for the automated proofs.

4.2 RRL

The RRL term rewriting system[KZ87] was used to perform the proof of satisfaction of the verification requirements. This system allows Knuth-Bendix completion of a set of rules and proof by rewriting. The completion facility was used in earlier work[Kir91] to produce a *complete*, i.e. confluent and terminating, set of rules for a subset of LOTOS operators. This rule set is embedded in Set 1 below.

There follows a brief description of the rules used; a more detailed discussion of the rules used in the system can be found in appendix B of this document.

The rules are divided into three sets:

- Set 1 contains rules derived from the weak bisimulation congruence laws, including the **hide** expansion law, which are given in appendix B.2.2 of the LOTOS standard[ISO88]. This set removes instances of the **hide** operator (by converting events to be hidden into the internal event, **i**, and reduces terms with respect to weak bisimulation congruence.
- Set 2 contains rules which "implement" the expansion law for parallelism, also found in appendix B.2.2 of the LOTOS standard. This rule set removes instances of the parallel operator, converting the terms into equivalent ones which use only sequencing and choice operators.
- Set 3 contains one of the **cred** laws, which can be found in appendix B.3.2 of the LOTOS standard. Set 3, when used, is always an addition to Set 1, since one of the laws for **cred** states that all the laws of weak bisimulation congruence are also true for **cred**. This allows terms to be further reduced with respect to the **cred** refinement relation.

Sets 1 and 2 also contain a core of basic rules on which the others build, e.g. rules for lists, sets and so on.

The aim of our experiments is to use these sets of rules to prove weak bisimulation congruence or testing congruence between the specification and the implementation of the system. Normally in a term rewriting system two terms, l and r, are shown to be equivalent by reducing them until two terms, l' and r', are obtained which are syntactically equivalent. If l' and r' are not syntactically equivalent and can be reduced no further then we may deduce that l and r are not equivalent. This procedure relies on the rule set used being complete. Our rule sets are not complete so our proofs are a little different.

If two terms can be shown to be equivalent by our rules, then they are also equivalent in the LOTOS semantics, but if two terms cannot be shown to be equivalent by our rules, then they may or may not be equivalent in the semantics. This is important in this case study, as a number of the proofs which were attempted could not be proved by RRL with our rules, and so the proofs had to be completed by hand.

Another side effect of having rule sets which are not complete is that the order in which the rules are applied is important. This is why the rules are split into three sets. In particular, if Set 2 is used then the reduction with respect to Set 2 must terminate before any rules form the other sets can be applied.

The next section contains details of the proofs which were attempted, and some discussion of why some of those proofs failed.

5 Proofs

In section 3 two possible approaches to proving that the implementation of the system satisfies the specification of the system were discussed. The first involved splitting the proof into three parts corresponding to the three protocols in the specification, while the second dealt with the system as a whole. The results of these approaches to trying to prove, using RRL, that the specification is in some way related to the implementation are given below.

5.1 Splitting the Proofs into Three Sections

Since each protocol describes the interface between two of the processes, the idea of proving each interface is correct, and deducing from that that the whole system is correct, is very appealing. Unfortunately, this approach turns out to be not as successful as might have been hoped. Proofs about the relationship between the specification and the implementation are still possible, but the results are not strong enough to show that the verification requirement is satisfied. However, examining the proofs which can be completed helps to discover the reasons for the failure of the approach overall. The proofs are categorised by the "satisfies" relation used, starting with the stronger of the two equivalences considered.

Weak Bisimulation Congruence To show that the verification requirement is satisfied the following equations must be shown to hold.

$$
\begin{aligned}
P1 \quad &\equiv_{wbc} \quad \textbf{hide}\,[m3, p3, n3, m4, p4, n4, m5, p5, m6, p6, m7, p7]\ \textbf{in} \\
&\qquad (A\ |[m1, p1, n1]|\ B) && (5) \\
P2 \quad &\equiv_{wbc} \quad \textbf{hide}\,[m1, p1, n1, m4, p4, n4, m5, p5, m7, p7]\ \textbf{in} \\
&\qquad (C\ |[m3, p3, n3, m6, p6]|\ B) && (6) \\
P3 \quad &\equiv_{wbc} \quad \textbf{hide}\,[m1, p1, n1, m3, p3, n3, m6, p6]\ \textbf{in} \\
&\qquad (D\ |[m4, p4, n4, m5, p5, m7, p7]|\ B) && (7)
\end{aligned}
$$

where $P1, P2, P3$ and A, B, C, D are as defined in section 2.

These equations cannot be proved using RRL. As described in section 4.2 there are two possible reasons for this: the first is that, as our rule sets are not complete, equations cannot always be shown to hold, even if they hold in the semantics. The second possibility is that the equations do not hold in the semantics. It can be shown that the latter is the case. The proof of this is given in appendix A.

Since the equations do not hold for weak bisimulation congruence a weaker relation must be considered.

Testing Congruence To show that the above equations hold for $\equiv_{tc}$, testing congruence, instead of $\equiv_{wbc}$, we can try to show that the equations hold in both directions for **cred**, i.e.

$$
\begin{aligned}
P1 \quad &\textbf{cred} \quad \textbf{hide}\,[m3, p3, n3, m4, p4, n4, m5, p5, m6, p6, m7, p7]\ \textbf{in} \\
&\qquad (A\ |[m1, p1, n1]|\ B) && (8) \\
P2 \quad &\textbf{cred} \quad \textbf{hide}\,[m1, p1, n1, m4, p4, n4, m5, p5, m7, p7]\ \textbf{in} \\
&\qquad (C\ |[m3, p3, n3, m6, p6]|\ B) && (9) \\
P3 \quad &\textbf{cred} \quad \textbf{hide}\,[m1, p1, n1, m3, p3, n3, m6, p6]\ \textbf{in} \\
&\qquad (D\ |[m4, p4, n4, m5, p5, m7, p7]|\ B) && (10)
\end{aligned}
$$

and vice versa. Note that the left and right hand sides of the equations above are exactly the same as those in equations (5), (6) and (7); only the relation in the middle has changed.

The proofs of equations (8) and (9) can be completed by RRL in the left-to-right direction, which means that the protocols are a deterministic reduction of the processes, i.e. the processes may have some nondeterminism not present in the protocols.

The proof of testing congruence should be completed by showing that the equations hold from right-to-left also, i.e. processes **cred** protocols. This cannot be done using RRL since the protocols are already in normal form with respect to the rule sets and it would be incorrect to proceed by reducing the term on the left hand side of the **cred** relation. Again, the proof has failed because these equations do not hold in the semantics. The proof of this is given in appendix A.

Equation (10) cannot be proved by RRL in either direction. The proofs that this equation does not hold in either direction in the semantics are also given in appendix A.

At this stage, it would appear that trying to prove that the verification requirements are satisfied is hopeless. However, we strongly believe that the processes are a valid implementation of the system and that therefore it is the approach to the proof which is incorrect. The strategy of splitting the proof into three parts does not work, or rather, some proofs can be completed, but they are not sufficient to prove that the verification requirement is satisfied. The next step is to try the alternate approach.

5.2 Proving the System as a Whole

At this stage no equivalences between the processes all combined and the protocols all combined can be demonstrated because although the processes can be combined using parallelism, there is no meaningful way in which to combine the protocols.

There are two operators which could be used to combine the protocols. These are sequential composition and interleaving (rather than general parallelism as the protocols have no events in common). The former cannot be used because, for example, the events of $P1$ do not all temporally precede the events of $P2$, and the latter cannot be used because the protocols contain no information about the relative ordering of events in different protocols. Interleaving results in a huge number of traces, many of which are meaningless in our example.

The reason the protocols cannot be successfully combined is that there is some information missing from the specification, leaving too large a step between it and the implementation. This information, which is implicit in the implementation, includes details of the timer, deallocation and what constitutes success or failure of the transaction. In the specification there is no information about any of these things. A solution to this problem is to add the information in the form of *constraints*, which gives a final, successful approach to the problem.

6 Adding Constraints to the Example

As stated above, the problem with the above approach is that some information, which appears in the implementation, is missing from the specification. All the extra information which was added to the implementation should somehow be added to the specification. This can be done by adopting the "constraint-oriented" method of specification[VSvS88].

This specification style relies on the way in which the LOTOS general parallelism operator handles multi-way synchronisation, i.e. synchronisation between two or more actions. For example, if three LOTOS processes, all of which use the action a, are combined using general parallelism, synchronising on a, then all three must perform that event at the same time. This means that different process expressions can specify different aspects of a behaviour, interacting to give the whole specification. The effect is similar to using conjunction in a logical specification; each part must be satisfied for the whole to be satisfied.

Using this method, more LOTOS processes are defined which express other aspects of the specification. The following extra information could be gleaned from the implementation.

- A timer is used by B to determine how long it should wait for D to send the *m5* message.
- Deallocation is compulsory rather than optional.
- Events occur in a particular order (mentioned in the informal overview of the system).
- Success or failure of the transaction as a whole, i.e. whether B replies *p1* or *n1* to A, relies on certain conditions.

The constraints to be added to the specification are as follows:

Timer Constraints

```
process timer :=   exit
                   [] set; (  tcancel; exit
                            [] timeout; exit)
                   endproc

process timer_on :=   exit
                      [] p4; set; exit
                      endproc

process timer_off :=   exit
                       [] set; (m5; tcancel; p5; m7; exit
                                [] timeout; m7; exit)
                       endproc
```

```
Deallocation Constraints
process dealloc_C :=   p3; m6; p6; exit
                     [] n3; exit
                     endproc

process dealloc_D :=   exit
                     [] m4; (  p4; m7; p7; exit
                            [] n4; exit)
                     endproc

Success and Failure
process system :=   m5; p1; exit
                  [] n3; n1; exit
                  [] n4; n1; exit
                  [] timeout; n1; exit
                  endproc

Ordering Constraints
process order13 := m1; m3; (  n3; n1; exit
                           [] p3; ( n1; exit
                                  [] p1; exit))
                   endproc
process order34 :=   m3; ( n3; n1; exit
                         [] p3; m4 (  n4; n1; exit
                                   [] p4; ( n1; exit
                                          [] p1; exit)))
                     endproc

process order457 :=   n3; n1; exit
                    [] m4; (  n4; n1; exit
                           [] p4; (  m5; p5; m7; p7; p1; exit
                                  [] timeout; m7; p7; n1; exit))
                    endproc

process order56 :=   n3; n1; exit
                   [] n4; m6; p6; n1; exit
                   [] timeout; m6; p6; n1; exit
                   [] m5; p5; m6; p6; p1; exit
                   endproc

process order67 :=   n3; n1; exit
                   [] p3; (  n4; m6; p6; n1; exit
                          [] p4; m6; p6; m7; p7; ( n1; exit
                                                 [] p1; exit))
                   endproc
```

As with the descriptions of the protocols and the processes, some "skip" branches are introduced to express the notion that a constraint may or may not be activated.

In the equation below, each protocol has as much information added to it as possible before combining it with the others. This is because in performing the proof, our system can only deal with one parallel statement at a time, which means that the proof has to be built up gradually from small units. Adding as much information as early as possible helps to cut down the size of the intermediate terms in the proof. The equation to be proved by RRL is:

$$
\begin{array}{l}
(((\mathrm{P1}\ |[p1, n1]|\ \mathrm{system})\ |[m1, p1, n1, n3]|\ \mathrm{order13}) \\
\quad |[p1, n1, m3, p3, n3, n4, m5, timeout]| \\
((((\mathrm{P2}\ |[p3, n3, m6, p6]|\ \mathrm{dealloc\_C}) \\
\quad |[m3, p3, n3, m6, p6]|\ (\mathrm{order34}\ |[p3, n3, p4, n4]|\ \mathrm{order67})) \\
\quad |[p1, n1, n3, m4, p4, n4, m7, p7]| \\
(((\mathrm{P3}\ |[m4, p4, n4, m7, p7]|\ \mathrm{dealloc\_D})\ |[m4, p4, n4, m5, p5, m7, p7]|\ \mathrm{order457}) \\
\quad |[p4, m5, p5, m7, timeout]| \\
((\mathrm{timer}\ |[set]|\ \mathrm{timer\_on})\ |[set, timeout, tcancel]|\ \mathrm{timer\_off})) \\
\quad |[m5, p5, m6, p6, timeout]|\ \mathrm{order56})) \\
\equiv_{wbc} \\
(((\mathrm{A}\ |[m1, p1, n1]|\ \mathrm{B})\ |[m3, p3, n3, m6, p6]|\ \mathrm{C}) \\
\quad |[m4, p4, n4, m5, p5, m7, p7]|\ \mathrm{D})
\end{array}
$$

This equation can be proved to hold by RRL. This is an adequate proof of the verification requirement since it means not only that the processes have the same observable behaviour as the protocols, but also that they behave in the same way in all contexts. The proof requires only the rules from set 2; no other rule sets are required. As the specification and implementation use no internal actions we may also deduce that the above equation holds for strong equivalence as well as for weak bisimulation congruence. As a bonus, since the constraints include deallocation information, the second verification requirement, that all the resources are correctly deallocated, is also satisfied.

6.1 Splitting the Proof (Revisited)

Due to the way in which the constraints operate, the proof can no longer be split into parts corresponding to the protocols. This is because some constraints, e.g. the ordering constraints, deal with events from more than one protocol.

To return to the failure of the original attempt to perform the proof in three parts, it now seems obvious why none of the proofs of equivalence could be completed. The implementation contains more information than the specification, and therefore certain events have to be "hidden". Hiding these events in the processes makes the nondeterminism present more obvious. In execution it is possible that the "wrong" branch of a choice could be taken, resulting in deadlock.

7 Extensions to the Example

The example as considered in this document is very simple; there are a number of ways in which it could be made more complex.

A useful extension would be to add an "abort" message, call it *m2*. A can abort the service at any time by sending *m2* to B, which should clean up by deallocating any resources held and then replying to A with *p2*.

In LOTOS it would be simple to add *m2 p2* as an abort sequence using the operator [> , which allows one process to take control from another. However, in this example the system is more complicated, requiring varying sequences of actions between *m2* and *p2*, depending on the events which occurred before *m2*. The original solution could not be easily extended to include this new behaviour. This could indicate a fault in the solution to the original problem, perhaps it is not modular enough, or it could be that there is no simple, elegant way to extend the solution. Certainly it is true to say that some of the constraints used in the final description and proof of the example are not perhaps the most obvious descriptions. In particular, some constraints contain too much information, in that they are not as modular as their names suggest. This information was necessary to be able to finish the proof. We do not claim that our solution is in any way optimal. Further investigation may reveal better solutions which can be easily extended when the specification of the problem is altered.

Other extensions to the example include the addition of data types to the messages, utilising the other part of the LOTOS description technique, extending B to allow it to re-send $m4$ an arbitrary number of times, or giving A the ability to send a "tell me your status" message to B at any time.

A more complicated extension would be the introduction of recursion to the language. As pointed out in section 2.2 this was avoided initially to make the problem simpler, but if the rule sets used in this case study are to have any application to real problems, the incorporation of rules to handle recursive operators is essential.

8 Review of the Tools Used

Although some degree of success was achieved in the case study, there were also many problems, not all of which arose from the example itself; some were due to either LOTOS or RRL.

8.1 Improvements to LOTOS

LOTOS was not always suitable to describe the example. A major problem was revealed when attempting to extend the original problem to include the abort message. The [> operator was unsuitable for this purpose because it does not allow the abort sequence to be dependent on the state of execution before the abort message. Our approach was to write each abort possibility into the LOTOS processes as choice branches, which made the specification rather cumbersome (and may have contributed to our inability to extend the original solution). An operator which allows the abort sequence to be flexible is required.

Another feature which would have been useful was an operator to "wrap-up" several actions and make them behave as a single action, i.e. like a critical section in a mutual exclusion problem. For example, we wanted to be able to combine two process expressions using interleaving, but to have a section in one of the expressions which, once it had begun, had to finish without interleaving with the other process until after the last action was completed. This could have been solved with a mutual exclusion algorithm, but a language construct to do this would be more convenient.

8.2 Improvements to RRL

The following is a list of features which we would have found useful in carrying out our proofs.

- The ability to specify which rule to apply next, particularly important when the rule set is not complete.
- A list of all the rules used in a reduction. RRL does not supply this information when a reduction is carried out.
- The ability to split rules into groups and allowing specification of which groups of rules could be used in a reduction. (We did this by running simultaneous copies of RRL, each with a different rule set. This requires the user to cut and paste equations from one system to another, which provides the opportunity to introduce errors.)
- The ability to reference the last term reduced and to use that reference in the next reduction. As our rule sets were not complete, the order in which rules were applied could be very important. For the case study proofs this meant that subparts of the proof had to be fully expanded before they could be added to the rest of the expression. Again we had to use cut and paste to perform this function, increasing the possibility of error.
- The ability to save rule sets, rather than having to regenerate them every time.
- Although RRL has fairly sophisticated functionality the user interface is poor, being only a list of commands followed by a prompt. A graphical interface would make the system more appealing to use.

Perhaps these features will be available in later versions of the software.

9 Conclusions and Further Work

After much experimentation, we succeeded in showing that the verification requirements of a small communications protocol were indeed satisfied by the given specification and implementation, although it must be noted that the given specification was not sufficient for our purposes and had to have more information added to enable the proofs to be carried out. The new information was added in a modular way however, and the text of the original specification was unaltered. This solution could not be easily extended to form a proof of the same requirements on a slightly modified system. Further work needs to be done here to discover why the solution is not easily extensible. It could be that there is a flaw in the way in which the problem is solved in the first place. Any new solutions to this example should be tested by trying to extend them to cope with alterations to the system. Suitable extensions to the problem are provided in section 7.

Acknowledgements

We would like to thank the members of the ERIL project group for many stimulating discussions on the issues and problems raised by this case study. In particular we would like to thank our colleagues at Glasgow, Muffy Thomas and Phil Watson, for their helpful comments on earlier versions of this work. We would also like to thank Deepak Kapur for letting us have RRL.

References

[BSS87] E. Brinksma, G. Scollo, and C. Steenbergen. LOTOS Specifications, their Implementations and their Tests. In B. Sarikaya and G.V. Bochmann, editors, *Protocol Specification, Testing, and Verification, VI*, pages 349–360. Elsevier Science Publishers B.V. (North-Holland), 1987.

[Dic90] A.J.J. Dick. A Case Study for the ERIL Project. Private communication, 1990.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[ISO88] International Organisation for Standardisation. *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1988.

[Kir91] C. Kirkwood. An Experiment using Term Rewriting Techniques for Concurrency. In S.L. Peyton-Jones, G. Hutton, and C.K. Holst, editors, *Functional Programming, Glasgow 1990*, pages 196–200. Springer-Verlag, 1991. Extended abstract.

[Kir92] C. Kirkwood. Using RRL for LOTOS. In preparation, 1992.

[KZ87] D. Kapur and H. Zhang. *RRL : Rewrite Rule Laboratory User's Manual*, 1987. Revised May 1989. Available by anonymous ftp from `herky.cs.uiowa.edu`.

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

[VSvS88] C.A. Vissers, G. Scollo, and M. van Sinderen. Architecture and Specification Style in Formal Descriptions of Distributed Systems. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification, VIII*, pages 189–204. Elsevier Science Publishers B.V. (North-Holland), 1988.

A Proofs omitted from the Text

In section 5 some equations were presented which could not be proved to hold using RRL. The proofs that these equations do not hold in the semantics of LOTOS are given here.

A.1 Definitions

We begin with some basic definitions.

- The domain of events is called A. This domain does not include the internal event $\mathbf{i}$. The set given by $A \cup \{\mathbf{i}\}$ is denoted Act. We use a to range over A, t to range over A^*, α to range over Act and σ to range over Act^*. The empty string is denoted ϵ.
- The domain of LOTOS processes, called $\mathcal{P}$, is the set of ground process expressions. We use p and q to range over $\mathcal{P}$.
- A labelled transition system is a 4-tuple $(S,\ Act,\ \{\xrightarrow{\alpha} \subseteq S \times S\}, s_0)$, which consists of a set S of *states*, a set Act of *transition labels*, a *transition relation* $\xrightarrow{\alpha}$, one for each $\alpha \in Act$, and a starting state $s_0 \in S$. The transition relation determines how we get from one state to another, and is defined by the inference rules which give an operational semantics of LOTOS. A LOTOS process is identified with the starting state of its labelled transition system by an abuse of notation.
- $p \xrightarrow{\alpha} q$ means that process p performs the event α and then behaves like q. This can be extended to strings: given $\sigma = m1\ m2\ m3$, $p \xrightarrow{\sigma} q$ is an abbreviation for $p \xrightarrow{m1} p' \xrightarrow{m2} p'' \xrightarrow{m3} q$.
- $p \overset{\alpha}{\Longrightarrow} q$ means that p performs some string of events $\mathbf{i}^j \alpha \mathbf{i}^k$ for some $j, k \geq 0$ before behaving like q. Again, this relation can be extended to strings as above.
- We write $\hat{\sigma}$ to denote the string σ with all occurrences of $\mathbf{i}$ removed.
- $p \overset{\hat{\alpha}}{\Longrightarrow} q$ means the same as $p \overset{\alpha}{\Longrightarrow} q$ for all $\alpha \neq \mathbf{i}$. For $\alpha = \mathbf{i}$ the behaviour is slightly different. Basically, $p \overset{\mathbf{i}}{\Longrightarrow} q$ means $p \xrightarrow{\sigma} q$ where $\sigma = \mathbf{i}^j$ for $j \geq 1$, while $p \overset{\hat{\mathbf{i}}}{\Longrightarrow} q$ denotes $p \xrightarrow{\hat{\sigma}} q$ where $\sigma = \mathbf{i}^j$ for $j \geq 0$, i.e. it is possible that $\sigma = \epsilon$.

Having made these definitions, we can now formally define weak bisimulation equivalence and congruence.

Definition 1 (Weak Bisimulation Equivalence) *A relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a weak bisimulation if $(p, q) \in R$ implies, $\forall \alpha \in Act$*

1. *if $\exists p' : p \xrightarrow{\alpha} p'$ then $\exists q' : q \overset{\hat{\alpha}}{\Longrightarrow} q'$ with $(p', q') \in R$, and*
2. *if $\exists q' : q \xrightarrow{\alpha} q'$ then $\exists p' : p \overset{\hat{\alpha}}{\Longrightarrow} p'$ with $(p', q') \in R$.*

Two process expressions, p and q, are weak bisimulation equivalent, written $p \approx q$, if $(p, q) \in R$ for some weak bisimulation R.

Definition 2 (Weak Bisimulation Congruence) *p and q are weak bisimulation congruent, $p \equiv_{wbc} q$, if $\forall \alpha \in Act$*

1. *if $\exists p' : p \xrightarrow{\alpha} p'$ then $\exists q' : q \overset{\alpha}{\Longrightarrow} q'$ with $p' \approx q'$, and*
2. *if $\exists q' : q \xrightarrow{\alpha} q'$ then $\exists p' : p \overset{\alpha}{\Longrightarrow} p'$ and $p' \approx q'$.*

Note that in this definition $\overset{\alpha}{\Longrightarrow}$ is used, whereas the definition of weak bisimulation equivalence $\overset{\hat{\alpha}}{\Longrightarrow}$ is used.

To define the **cred** relation some more auxilliary definitions are required.

- p **after** $t = \{p' | p \stackrel{t}{\Longrightarrow} p'\}$.

- Let $L \subseteq A$, then (p **must** L) iff. $p \stackrel{\epsilon}{\Longrightarrow} p'$ implies that there exists a p'' with $p' \stackrel{l}{\Longrightarrow} p''$ for some $l \in L$. The first part ensures that p is not divergent, the second says that the only (visible) actions p may perform are those in L.

- The **must** predicate can be defined over sets of states in the obvious way. Let $\mathcal{Q} \subseteq \mathcal{P}$, then $\mathcal{Q}$ **must** L iff. q **must** L for all $q \in \mathcal{Q}$.

- Let $C[\,]$ be a LOTOS context, then $C[p]$ denotes the process p in the context $C[\,]$.

The **red** and **cred** relations may now be defined.

Definition 3 (The red relation) p **red** q *iff.*
$\forall t \in A^* \ \forall L \subseteq A.(q$ **after** $t)$ **must** L *implies* $(p$ **after** $t)$ **must** L

Definition 4 (The cred relation) *For LOTOS behaviour expressions p and q, p* **cred** *q iff. for all LOTOS contexts $C[\,]$, $C[p]$* **red** *$C[q]$.*

Both of these refinement relations are preorders (i.e. reflexive and transitive relations).

A.2 Proofs

Note that the equations to be proved in this section are all of the form:

$$\begin{aligned} P1 &= \textbf{hide}\ [m3, p3, n3, m4, p4, n4, m5, p5, m6, p6, m7, p7]\ \textbf{in} \\ &\quad (A\ |[m1, p1, n1]|\ B) \qquad (11) \\ P2 &= \textbf{hide}\ [m1, p1, n1, m4, p4, n4, m5, p5, m7, p7]\ \textbf{in} \\ &\quad (C\ |[m3, p3, n3, m6, p6]|\ B) \qquad (12) \\ P3 &= \textbf{hide}\ [m1, p1, n1, m3, p3, n3, m6, p6]\ \textbf{in} \\ &\quad (D\ |[m4, p4, n4, m5, p5, m7, p7]|\ B) \qquad (13) \end{aligned}$$

where the only parts which change from one proof to another is the relation substituted for $=$, and the orientation in the case of the **cred** refinement relation. In the following proofs, we use $P1, P2$ and $P3$ to denote the left hand sides of equations (11), (12) and (13) respectively, and *procAB, procCB* and *procDB* to denote the right hand sides of equations (11), (12) and (13) respectively.

A.2.1 Weak Bisimulation Congruence

The first proof which has to be demonstrated is that the equations given above do not hold for weak bisimulation congruence, i.e.

$$\begin{aligned} &not\ (P1 \ \equiv_{wbc} \ procAB) \qquad (14) \\ &not\ (P2 \ \equiv_{wbc} \ procCB) \qquad (15) \\ &not\ (P3 \ \equiv_{wbc} \ procDB) \qquad (16) \end{aligned}$$

Informally, the difference between weak bisimulation equivalence and weak bisimulation congruence lies in the way in which initial internal actions are treated. Let p denote the left hand side of an equation and q the right hand side, as in definitions 1 and 2. In both the equivalence and the congruence every action performed by p must be matched by an action performed by q, where extra internal actions may be added in the matching process. Similarly, every action performed by q must be matched by an action performed by p. However, for weak bisimulation congruence, an initial internal action must be matched by one or more internal actions, while weak bisimulation equivalence allows an initial internal action to be matched by zero or more internal actions. This is the key to the proof.

$P1$ **normal form (wbc) is**

m1; ((n1; exit) [] (p1; exit))

$P2$ **normal form (wbc) is**

m3; ((n3; exit) [] (p3; (exit [] (m6; p6; exit))))

$P3$ **normal form (wbc) is**

```
  exit
[] m4; (n4; exit
      [] p4; (exit
             [] m7; p7; (exit [] m5; p5; exit)
             [] m5; p5; (exit [] m7; p7; exit)))
```

procAB **normal form (wbc) is**

m1; ((**i**; n1; exit) [] (**i**; p1; exit))

procCB **normal form (wbc) is**

i; m3; (n3; exit [] p3; m6; p6; exit)

procDB **normal form (wbc) is**

```
i; (i; exit
  [] i; m4; (n4; exit
            [] p4; (i; m7; p7; exit
                  [] m5; p5; m7; p7; exit)))
```

Figure 2: Normal Forms of the Processes and the Protocols

The actions which can be performed by the various components of the system can be more easily seen by examining normal forms with respect to $\equiv_{wbc}$ produced by RRL of the terms in equations (14), (15) and (16). These are given in figure 2. Note that the protcols have not been reduced — they were already in normal form with respect to our rule sets.

The normal forms in figure 2 will be referred to throughout this section. Normally we would be able to deduce that equations (14), (15) and (16) do not hold because the normal forms for the left and right hand sides are not syntactically the same. However, since our rule sets are not complete, the terms given in figure 2 are not necessarily the *unique* normal forms, they are only the normal forms with respect to our rules. This means we must prove the equations do not hold by referring to the semantics of LOTOS.

Informally stated, the proof proceeds as follows for each equation:

Equation (14) Clause (1) of definition (2) holds; *procAB* can match every action $P1$ can perform by a sequence of actions. However, clause (2) fails because (*procAB* **after** *m1*) can perform an **i** action to get into a situation in which it can perform the action *n1* and no other, whereas ($P1$ **after** *m1*) can only match an **i** action by doing nothing, which leaves it in a state in which either *n1* or *p1* are possible. Obviously a state in which two distinct actions are possible cannot be equivalent to one in which only one action is possible.

Equation (15) Again clause (1) is satisfied, but clause (2) fails as weak bisimulation congruence insists that an initial internal action is matched by one or more internal actions. $P2$ cannot match the initial internal action performed by *procCB* except by zero actions, and therefore the expressions are not weak bisimulation congruent.

Equation (16) Exactly the same as for equation (15).

This ends the proof.

A.2.2 Testing Equivalence

We now go on to proofs involving the refinement relation **cred**. The first proof is that the equations presented at the start of this section do not hold in the right-to-left direction for the **cred** relation, i.e. we wish to prove

$$\mathit{not}\ (\mathit{procAB} \quad \mathbf{cred} \quad P1) \tag{17}$$
$$\mathit{not}\ (\mathit{procCB} \quad \mathbf{cred} \quad P2) \tag{18}$$
$$\mathit{not}\ (\mathit{procDB} \quad \mathbf{cred} \quad P3) \tag{19}$$

In definition 4, the **cred** relation was characterised by the tests a process must pass. To prove that two LOTOS processes are not related by **cred**, a test must be found which one must pass but the other need not. In particular, because the orientation of the equation matters, a test must be found which the right hand side must pass, but which the left hand side may fail. For example, in equation (17) a test must be found which $P1$ must pass, but which *procAB* may fail. This is sufficient to prove that the **cred** relation does not hold in the given direction. If no such test can be found, there may be a context which differentiates the terms, since **cred** is a congruence relation.

In the following proofs we refer back to the normal forms given for weak bisimulation congruence in figure 2 and use these in the proofs. This procedure is sound.

Proof. To make use of the normal forms we need to show

$$\mathit{not}\ (p' \ \mathbf{cred}\ \ q') \Rightarrow \ \mathit{not}\ (p \ \mathbf{cred}\ \ q) \text{ where } p \ \equiv_{wbc}\ p' \text{ and } q \ \equiv_{wbc}\ q'$$

i.e. p' and q' denote the normal forms of p and q respectively.

The method of proof is by contradiction. Our assumptions are: $p \ \equiv_{wbc}\ p'$ and $q \ \equiv_{wbc}\ q'$, which we can show using RRL, and *not* (p' **cred** q'), which we will demonstrate for each case in the remainder of this section. Now suppose p **cred** q. Since $\vdash (p \ \equiv_{wbc}\ q) \ \Rightarrow (p \ \mathbf{cred}\ q)$, from [ISO88], we may deduce p' **cred** p and q **cred** q', and hence p' **cred** q' by transitivity of **cred**. This is a contradiction. ∎

Although there are **cred** laws which would reduce some of the normal forms given further, they are not used here as they apply only to the *proc* normal forms and reducing terms on the left hand side of the **cred** relation when we are trying to show that it doesn't hold would lead to false conclusions.

Again informally stated, the proofs that equations (17), (18) and (19) hold are as follows:

Equation (17) ($P1$ **after** *m1*) can always pass the test *n1*, but (*procAB* **after** *m1*) will sometimes fail that test (because the internal actions can proceed silently and put us into a state in which only *p1* can go ahead).

Equation (18) There is no test which can differentiate between $P2$ and *procCB*, however, they can be differentiated by the context *m8* [] p, where *m8* is an arbitrarily chosen action and p is a place holder for the system under test. In this context ($P2$ **after** ϵ) can always pass the test *m8*, whereas (*procCB* **after** ϵ) could fail this test. This is because the internal action can move *procCB* into a state where only *m3* is possible.

Equation (19) Given the test *m5*, ($P3$ **after** *m4 p4 m7 p7*) must always pass this test, but (*procDB* **after** *m4 p4 m7 p7*) can never pass this test.

This concludes the proof.

The final proof which must be carried out is that equation (10) of section 5.1 does not hold, i.e.

$$\mathit{not}\ (P3 \quad \mathbf{cred} \quad \mathit{procDB}) \tag{20}$$

The proof proceeds by demonstrating that there is a test which *procDB* must satisfy which may not be satisfied by $P3$. Consider the following:

($procDB$ **after** $m4\ p4\ m7\ p7\ m5$) **must** L for any $L \subseteq A$, since ($procDB$ **after** $m4\ p4\ m7\ p7\ m5$) $= \emptyset$ and $\emptyset$ **must** L is vacuously true. However, ($P3$ **after** $m4\ p4\ m7\ p7\ m5$) **must** L is only true of sets L such that $p5 \in L$. So, in particular, if $L = \{m1\}$ then ($procDB$ **after** $m4\ p4\ m7\ p7\ m5$) **must** L, but *not* (($P3$ **after** $m4\ p4\ m7\ p7\ m5$) **must** L). $P3$ is not a refinement of $procDB$.

This concludes the proof that equation (20) holds, and also this collection of proofs.

B LOTOS Laws

This appendix gives a more detailed description of the rule sets used in RRL for this case study.

As described in section 4.2 there are three sets of rules being used for the proofs in this case study. Common to the first two rule sets is a core of basic rules, i.e. rules for sets, lists, generalised choice (choice over sets of processes, denoted $[]\,\{B_i\}$ where each B_i is a process) and constants (each constant must be declared to be unique — this is done by mapping the event constants into the natural numbers). These rules are not discussed here; a more extensive discussion of the way in which the rule sets were built up, and why particular rules were chosen, can be found in [Kir92].

In addition to these core rules, set 1 contains some rules derived from the weak bisimulation congruence laws found in appendix B.2.2 of the LOTOS standard. This rule set contains some rules which are unlikely to be used in this particular example, they are mentioned for completeness. Using the numbering system of the LOTOS standard, the laws in the rule set are: b1, b2, b3, b4, c1, c3a, c3b, c7, d1, d2a, d3, e2, e3, e4, e5, m1, m2 and m3. The following are those rules which are relevant to this example.

$$\begin{array}{lll} P\,[]\,Q & \equiv_{wbc} & Q\,[]\,P \\ (P\,[]\,Q)\,[]\,R & \equiv_{wbc} & P\,[]\,(Q\,[]\,R) \\ P\,[]\,\mathbf{stop} & \equiv_{wbc} & P \\ P\,[]\,P & \equiv_{wbc} & P \end{array} \qquad \begin{array}{r} (b1) \\ (b2) \\ (b3) \\ (b4) \end{array}$$

$$\begin{array}{lll} P \mid Q & \equiv_{wbc} & Q \mid P \\ \mathbf{exit} \mid \mathbf{exit} & \equiv_{wbc} & \mathbf{exit} \\ \mathbf{exit} \mid \mathbf{stop} & \equiv_{wbc} & \mathbf{stop} \end{array} \qquad \begin{array}{r} (c1) \\ (c3a) \\ (c3b) \end{array}$$

$$\begin{array}{lll} a; \mathbf{i}; P & \equiv_{wbc} & a; P \\ P\,[]\,\mathbf{i}; P & \equiv_{wbc} & \mathbf{i}; P \\ a; (P\,[]\,\mathbf{i}; Q)\,[]\,a; Q & \equiv_{wbc} & a; (P\,[]\,\mathbf{i}; Q) \end{array} \qquad \begin{array}{r} (m1) \\ (m2) \\ (m3) \end{array}$$

In addition to these simple rules, set 1 also contains the expansion law for **hide** which propagates the application of **hide** through the LOTOS process, performing the conversion of hidden events to **i**.

$$\begin{array}{lll} \mathbf{hide}\ \mathrm{A}\ \mathbf{in}\ \mathrm{B} & = & []\,\{b_i;\ \mathbf{hide}\ \mathrm{A}\ \mathbf{in}\ B_i \mid name\,(b_i) \notin \mathrm{A},\ \mathrm{i} \in \mathrm{I}\} \\ & & [][]\,\{\mathbf{i};\ \mathbf{hide}\ \mathrm{A}\ \mathbf{in}\ B_i \mid name\,(b_i) \in \mathrm{A},\ \mathrm{i} \in \mathrm{I}\} \end{array}$$

where A is some list of events,

B $= []\,\{b_i; B_i \mid i \in I\}$,

and *name* extracts the gate name from an event.

Set 2 contains rules which implement the expansion law for parallelism.

$$\begin{array}{lll} \mathrm{B} \mid[A]\mid \mathrm{C} = & []\,\{b_i;\ (B_i \mid[A]\mid \mathrm{C}) \mid name\,(b_i) \notin A, i \in I\} \\ & [][]\,\{c_j;\ (\mathrm{B} \mid[A]\mid C_j) \mid name\,(c_j) \notin A, j \in J\} \\ & [][]\,\{\mathrm{a};\ (B_i \mid[A]\mid C_j) \mid \mathrm{a} = b_i = c_j,\ name\,(\mathrm{a}) \in A, i \in I, j \in J\} \end{array}$$

where B $= []\,\{b_i; B_i \mid i \in I\}$,

C $= []\,\{c_j; Cj \mid j \in J\}$,

and *name* extracts the gate name from an event.

This says that in an interleaving model of parallelism, there must be some notion of which actions are performed first. This law gives us all the possible alternatives.

Previously specific instances of this law tailored to the problem under consideration have been generated by hand, but in this case study far too many are needed and it is unreasonable to continue to use this method. Instead, a large number of rules have been added to the system which "implement" this expansion law in a recursive manner. These have been tested extensively for correctness, but have yet to be *proved* correct. This rule set also contains some subsidiary operators and associated rules which are needed to implement the expansion law.

Set 3 contains rules derived from the laws relating to the **cred** relation, found in appendix B.3.2. The laws used from B.3.2 are 1 and 2.

$$\begin{array}{rcl} \vdash (P \equiv_{wbc} Q) & \textit{implies} & (P \ \mathbf{cred} \ Q) \\ P & \mathbf{cred} & \mathbf{i}; P \end{array}$$

Law (1) says that **cred** inherits all the laws of weak bisimulation congruence, which we make use of by using set 3 in conjunction with set 1. Law (2) says that occurrences of the internal event may be removed, no matter where they occur. Contrast this with the weak bisimulation congruence law ($m1$) which allows internal events to be removed if they come in between an event and a process. Essentially this results in all "leading" internal events being left behind, i.e. if an internal action contributes to a nondeterministic choice by being the first event in a process expression then it is left in.