On the Mechanized Verification of Infinite Systems

Christine Röckl and Javier Esparza
Fakultät für Informatik, Technische Universität München
D-80290 München, Germany
http://www7.in.tum.de/{~esparza,~roeckl}

Abstract

Observation equivalence is a well-known technique for proving that a concurrent system satisfies its specification. We report on our experience in the mechanization of observation equivalence proofs with the help of a general-purpose theorem prover. Several case-studies are considered, incluiding an sliding window and a cache-coherence protocol. In all cases the system has an infinite number of states, and sometimes also an arbitrarily large number of components. We show how compositionality and bisimulation-up-to techniques can be applied to reduce the size of the proofs.

1 Introduction

One of the main goals of the A3 Teilprojekt within the Sonderforschungsbereich 342 is the development of verification techniques for distributed systems. Since 1995, special emphasis has been put on automatic (model-checking) and semi-automatic techniques, and in the period 1998-2000, A3 has devoted special attention and resources to the development of machine support for the analysis of systems with an *infinite* number of states. Dealing with infinite-state systems is one of the keys to extending the success of formal verification from hardware to software.

The A3 Teilprojekt has designed fully automatic verification algorithms for restricted classes of systems exhibiting a regular structure. Since these results cannot be extended to arbitrary systems due to well-known undecidability results, A3 has also investigated the use of *interactive theorem*

provers. The goal is the mechanization of a proof technique allowing to split a correctness proof into a potentially large number of "easy" cases, and a small number of "difficult" cases. The theorem prover should be able to automatically solve (most of) the easy cases, and guarantee that the proof of the difficult cases, carried out interactively with humans, is correct and complete. Moreover, the proof technique should offer the possibility to divide the correctness proof of a system into proofs about their subsystems. That is, the proof technique should exhibit good *compositionality* properties.

A3 has chosen *observational equivalence*, as introduced by Milner and Park in classical papers [11, 6], as proof technique. In this approach, both the system and its specification are formalized either as processes Sy and Sp in an adequate process algebra, or simply as infinite labelled transition systems. The system satisfies the specification if Sy and Sp are *observationally equivalent*, meaning intuitively that their behaviours cannot be distinguished by an external observer.

Observational equivalence can be shown by exhibiting a relation between the possible states of Sy and Sp, and proving that this relation is a *bisimulation*. Loosely speaking, checking that a given relation is a bisimulation amounts to proving for each pair (s, t) in the relation and for each possible action of state s (t) the existence of a "matching" action of state t (s). The proof is carried out by partitioning the relation into a suitable number of subrelations for whose elements the matching property can be proved in an analogous way. For most of these subrelations the proof is simple, and can be done automatically or with little interaction.

In this paper, we apply the technique of observational equivalence to a number of systems having an infinite state space. Proofs are carried out in the theorem prover Isabelle/HOL. In Section 3 we present two introductory examples which allow us to demonstrate how to exhibit bisimulations in a theorem prover. The second of these examples is the well-known alternating-bit protocol. In Section 4 we consider a sliding window protocol. Correctness is proved for a window of arbitrary size n. The proof exploits the compositionality of observation equivalence. Finally, in Section 5 we consider a write-invalidate cache coherence protocol. We show correctness using bisimulation-up-to techniques, which allow to substantially reduce the size of the proof.

2 Isabelle/HOL

We use the general purpose theorem prover Isabelle/HOL [13, 12]. Isabelle's *meta-logic* is higher-order intuitionistic logic. On top of it, various *object-*

logics are defined, including a theory of higher-order logic (HOL), which is the one we use. Users can extend Isabelle's object-logics with *theories* of their own; in particular, HOL allows the user to define recursive datatypes and inductive sets, from which Isabelle automatically computes proof strategies for structural and rule induction, respectively. A theory generally consists of two parts: in a so-called `.thy`-file, the user defines new types and objects, and in the associated `.ML`-file, he/she can formally derive *theorems* about the new objects.

Proofs in Isabelle are based on unification, and are usually conducted in a backward style: the user formulates the goal he/she intends to prove, and then—in interaction with Isabelle—continuously reduces it to simpler subgoals until all of the subgoals have been accepted by the tool. Upon this, the goal can be stored in the theorem database of Isabelle/HOL to be applicable in further proofs. The prover offers various tactics, most of them applying to single subgoals. The basic resolution tactic `resolve_tac`, for instance, allows the user to instantiate a theorem from Isabelle's database so that its conclusion can be applied to transform a current subgoal into instantiations of its premises. Besides these *classical tactics*, Isabelle offers *simplification tactics* based on algebraic transformations. Powerful *automatic tactics*, like `auto_tac` or `force_tac`, apply the basic tactics to prove given subgoals according to different heuristics.

3 Applying Observational Equivalence

We assume that the system and its specification are modelled in terms of labelled transition systems [14]. A *labelled transition system* consists of a set of transitions of the form $P \xrightarrow{\alpha} P'$, where P and P' are *processes* or *states*, and α is an *action*. There are *visible* actions, included in a set $\mathcal{A}ct$, which can be inputs $a(\tilde{v})$ (receive a list $\tilde{v}$ of values through channel a) or outputs $\bar{a}(\tilde{v})$ (send a list $\tilde{v}$ of values through channel a); and an *invisible* or *internal* action τ. We write a and $\bar{a}$ whenever the list of values is empty. There is also a distinguished *initial state*.

Processes can either be *basic* processes or a *parallel composition* of basic processes. The behaviour of a process P, that is, the transitions of the form $P \xrightarrow{\alpha} P'$, is defined inductively, as the *least* set of transitions satisfying given *introduction rules*. For a basic process P, these are *axioms* of the form $P \xrightarrow{\alpha} P'$, possibly with side conditions. For example, the parameterized set of axioms

$$A(x) \xrightarrow{\bar{a}(x \bmod 2)} A(x+1), \qquad x \in I\!N,$$

describes the transitions of the basic processes $A(0), A(1), \ldots$, yielding the infinite transition system

$$A(0) \xrightarrow{\bar{a}(0)} A(1) \xrightarrow{\bar{a}(1)} A(2) \xrightarrow{\bar{a}(0)} A(3) \ \ldots$$

Taking $A(0)$ as initial state, we have thus modelled a system which repeatedly outputs the values 0 and 1 through the channel a.
For a parallel composition of processes, the introduction rules are as follows:

- $P \xrightarrow{\alpha} P'$ implies $P \parallel Q \xrightarrow{\alpha} P' \parallel Q$,
- $Q \xrightarrow{\alpha} Q'$ implies $P \parallel Q \xrightarrow{\alpha} P \parallel Q'$,
- $P \xrightarrow{\bar{a}(\tilde{v})} P'$ and $Q \xrightarrow{a(\tilde{v})} Q'$ imply $P \parallel Q \xrightarrow{\tau} P' \parallel Q'$, and
- $P \xrightarrow{a(\tilde{v})} P'$ and $Q \xrightarrow{\bar{a}(\tilde{v})} Q'$ imply $P \parallel Q \xrightarrow{\tau} P' \parallel Q'$.

The inductiveness of these rules yields a case analysis telling for a transition how it must have been derived. For a visible transition $P \parallel Q \xrightarrow{\alpha} R$, for instance, we can infer that there must exist some P' or Q' such that $P \xrightarrow{\alpha} P'$ and $R \equiv P' \parallel Q$, or $Q \xrightarrow{\alpha} Q'$ and $R \equiv P \parallel Q'$. These inferences are automatically computed by Isabelle/HOL, and in our case studies we make extensive use of this feature.

Being interested in the visible behaviour of the systems rather than in their internal activities, we use *weak* transitions abstracting over τ-transitions. (In the sequel, we also refer to normal transitions as *strong* transitions.) We denote the reflexive transitive closure of $\xrightarrow{\tau}$ by $\stackrel{\epsilon}{\Longrightarrow} \stackrel{\text{def}}{=} (\xrightarrow{\tau})^*$, and $\stackrel{\alpha}{\Longrightarrow}$ is given by $\stackrel{\epsilon}{\Longrightarrow}\xrightarrow{\alpha}\stackrel{\epsilon}{\Longrightarrow}$. In particular, $P \xrightarrow{\alpha} P'$ implies $P \stackrel{\alpha}{\Longrightarrow} P'$.

Bisimulation-based equivalences have been introduced by Park and Milner [11, 6]. For technical reasons, we slightly modify the standard definition and use *bisimulations wrt. a set of visible actions*[1]. As usual, $\hat{\alpha}$ denotes α if α is a visible label, and ϵ if $\alpha \equiv \tau$.

Definition 1 (Bisimulation) *$\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is a* bisimulation wrt. *$L \subseteq \mathcal{A}ct$ if for all $P\mathcal{R}Q$, all $\alpha \in L \cup \{\tau\}$, all $P' \in \mathcal{S}_1$, and all $Q' \in \mathcal{S}_2$,*

[1]For readers with knowledge of process algebra, this more general definition makes the restriction operator unnecessary, and so we can avoid formalizing additional transition rules for processes with restriction.

(i) If $P \xrightarrow{\alpha} P'$, *there exists* $Q' \in \mathcal{S}_2$ *such that* $Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ *and* $P' \mathcal{R} Q'$.

(ii) If $Q \xrightarrow{\alpha} Q'$, *there exists* $P' \in \mathcal{S}_1$ *such that* $P \stackrel{\hat{\alpha}}{\Longrightarrow} P'$ *and* $P' \mathcal{R} Q'$.

Two processes P, Q *are* observationally equivalent wrt. L *if some bisimulation wrt.* L *contains* (P, Q). *Equivalently, if* $\approx_L$ *is defined as the union of all bisimulations wrt.* L, *then* P, Q *are observationally equivalent wrt.* L *if and only if* $P \approx_L Q$.

So, loosely speaking, if two processes are observationally equivalent, then any action of one of the processes can be matched by a sequence of actions of the other process, and the results are again observationally equivalent. This definition yields a method for a systematic analysis of reactive systems. In order to prove that a system *Sy* is observationally equivalent to its specification *Sp*, proceed as follows: (1) exhibit a relation containing the initial states of *Sy* and *Sp*, and (2) prove that it is a bisimulation.

We illustrate how this procedure is carried out in Isabelle/HOL by means of a toy example. Define a process B by

$$B \xrightarrow{\bar{a}(0)} B' \qquad\qquad B' \xrightarrow{\bar{a}(1)} B$$

We prove that the process $A(0)$, defined above, and B are observationally equivalent. In a first step, we set up the systems and their specifications by giving their states and transition rules in an inductive definition.

$$\begin{array}{ll} \textbf{A1} & A(x) \xrightarrow{\bar{a}(x \bmod 2)} A(x+1), \quad x \in I\!N \\ \textbf{B1} & B \xrightarrow{\bar{a}(0)} B' \\ \textbf{B2} & B' \xrightarrow{\bar{a}(1)} B \end{array}$$

From this definition, Isabelle computes sets of rules that can be used to reason about the transitions in a constructive as well as in an analysing style. For instance, Isabelle computes an *elimination* rule stating that, for every i, if $A(i)$ can perform a transition $A(i) \xrightarrow{\alpha} P'$ then necessarily $\alpha = \bar{a}(i \bmod 2)$ and $P' = A(i+1)$.

In our example, the bisimulation candidate is

$$\begin{array}{rcl} \mathcal{F} & \stackrel{\text{def}}{=} & \mathcal{F}_1 \cup \mathcal{F}_2 \\ \mathcal{F}_1 & \stackrel{\text{def}}{=} & \{(A(i), B) \mid i \text{ even}\} \\ \mathcal{F}_2 & \stackrel{\text{def}}{=} & \{(A(i), B') \mid i \text{ odd}\} \end{array}$$

In Isabelle, this relation is proved to be a bisimulation wrt. the set $\{\bar{a}(0), \bar{a}(1)\}$, by first using a (straightforward) formalization of Definition 1 as a rewrite rule, that is, by expanding the definition, and then deriving the resulting proof obligations using Isabelle's automatic tactic `auto_tac`.

Often, further theorems are necessary to provide additional information about data types and functions used to model the transition systems (for instance, as we shall see later, about insertion into or deletion from the finite lists representing communication channels). In our example we need the following basic theorems about the mod function:

$\mathbf{T1}_o$	$(2n+1) \bmod 2 = 1$	$\mathbf{T1}_e$	$2n \bmod 2 = 0$
$\mathbf{T2}_o$	$n \bmod 2 = 1 \Rightarrow \exists m.\, n = 2m+1$	$\mathbf{T2}_e$	$n \bmod 2 = 0 \Rightarrow \exists m.\, n = 2m$

In order to show that $\mathcal{F}$ is a bisimulation wrt. $\{\bar{a}(0), \bar{a}(1)\}$, we must prove the following property for each $i \in \{1, 2\}$ and for each $\alpha \in \{\bar{a}(0), \bar{a}(1), \tau\}$: for every $(P, Q) \in \mathcal{F}_i$, if $P \xrightarrow{\alpha} P'$, for some P', then there exists a Q' such that $Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{F}$; and similarly for Q. So even in this toy example, we have $2 \times 3 \times 2 = 12$ proof obligations to fulfill. Seven of them are carried out automatically by Isabelle using its tactic `auto_tac`. For the remaining five obligations, very moderate interaction by the user is necessary, mostly consisting of a suitable instantiation of results like $\mathbf{T1}_o$ through $\mathbf{T2}_e$.

Remark: It should be noted that, unlike in many proof mechanizations in theorem provers, our Isabelle proofs are not much different from proofs as one would perform them by hand. Yet, what is different is the emphasis put on different parts of the proofs. Whereas in proofs by hand one has to be careful not to forget any of the proof obligations, Isabelle automatically takes care of this. On the other hand, an Isabelle user may have to spend a lot of time interacting with the tool in order to prove simple theorems about the data structures manipulated during a transition. In both cases, the weak transitions have to be found by the person conducting the proof. However, presenting them to Isabelle is not much more time-consuming than writing them down on a piece of paper, and often it even suffices to provide the prover with a scheme that enables it to generate the transitions automatically.

Observational equivalence is a compositional proof technique. In our context, this means that if P and Q are observationally equivalent wrt. a set L of actions, then $P \parallel R$ is observationally equivalent to $Q \parallel R$ wrt. L for *any* process R. In other words, P can be replaced by Q in any context. We shall make frequent use of this property.

K1	$K(s) \xrightarrow{\mathrm{acc}(x)} K(x_c s)$	**L1**	$L(s) \xrightarrow{\mathrm{acc}(x)} L(xs)$	(accept)
K2	$K(sx_bt) \xrightarrow{\tau} K(st)$	**L2'**	$L(sxt) \xrightarrow{\tau} L(st)$	(lose)
K3	$K(sx_bt) \xrightarrow{\tau} K(sx_bx_bt)$	**L3**	$L(sxt) \xrightarrow{\tau} L(sxxt)$	(duplicate)
K4	$K(sx_bt) \xrightarrow{\tau} K(sx_gt)$			(garble)
K5	$K(sx_b) \xrightarrow{\overline{\mathrm{d}}(x_b)} K(s)$	**L5**	$L(sx) \xrightarrow{\overline{\mathrm{del}}(x)} L(s)$	(deliver)
F1	$F \xrightarrow{\mathrm{d}(x_c)} F(x)$	**F1'**	$F \xrightarrow{\mathrm{d}(x_g)} F$	(accept/discard)
F2	$F(x) \xrightarrow{\tau} F$	**F2'**	$F(x^{n+1}) \xrightarrow{\tau} F(x^n)$	(lose, $n > 0$)
F3	$F(x^n) \xrightarrow{\tau} F(x^{n+1})$			(duplicate)
F4	$F(x) \xrightarrow{\overline{\mathrm{del}}(x)} F$	**F4'**	$F(x^{n+1}) \xrightarrow{\overline{\mathrm{del}}(x^n)} F$	(deliver, $n > 0$)

Table 1: Faulty Communication Channels. An implementation K can lose, duplicate, and garble accepted messages (**K1**–**K5**). We assume garbling of messages to be detectable. A bit b attached to each message tells whether the message is still correct ($b = c$) or has been garbled ($b = g$). In combination with a filter F (**F6**–**F9**, **F6'**–**F9'**), it behaves like a specification L (**L1**–**L5**).

3.1 Faulty channels of unbounded size

Our first example is taken from [20]. It is of interest to us for the following reasons: (1) it compares two nondeterministic infinite-state systems operating on similar data structures, and (2) for most of the resulting proof obligations it suffices to find matching strong transitions, that is, in most cases an action by one of the processes is matched by one single action of the other processes. Reason (1) emphasizes that the bisimulation proof method is universally applicable, and does not require, for instance, the specification to be deterministic. As we shall see, reason (2) allows Isabelle/HOL to conduct the proof almost automatically.

Consider two channels, K and L, of unbounded capacity; their contents can be modelled by finite lists of arbitrary length, and so we denote a state of channel K by $K(x_1 \ldots x_n)$, where $x_1 \ldots x_n$ is a list of messages, each of them taken from an arbitrary set M of possible messages. K and L are basic processes, and Table 1 contains the formal descriptions of their behaviour. Both K and L may lose (**K2**, **L2**) or duplicate (**K3**, **L3**) messages, but K is further able to garble data (**K4**). We suppose that it is possible to detect whether a message is correct or garbled; therefore, all messages in K are marked with an index c if they are correct, and g if they are garbled; b stands for b or g. We consider a filter F which, when attached to a channel,

delivers correctly transmitted messages (**F1**, **F4**, **F4'**) and discards garbled ones (**F1'**). Like the channels, F is itself faulty: it can lose (**F2**, **F2'**) or duplicate (**F3**) messages. We prove that the parallel composition of K and F is observationally equivalent to L with respect to the set of actions $L_1 = \{\, \mathrm{acc}(x), \overline{\mathrm{del}}(x) \mid x \in M \,\}$. (This is not so trivial as it may seem: for instance, it does not hold if F cannot lose messages, see the remark below.)

For this, we show that the relation

$$\begin{array}{rcll} \mathcal{F} & \stackrel{\mathrm{def}}{=} & \{\, (K(s) \parallel F, \quad L(\hat{s})) & \mid s \text{ an indexed sequence of messages}, \\ & & & \quad \hat{s} = s \text{ without indices and garbage} \,\} \\ & \cup & \{\, (K(s) \parallel F(x^n),\ L(\hat{s}x^n)) & \mid s \text{ an indexed sequence of messages}, \\ & & & \quad \hat{s} = s \text{ without indices and garbage}, \\ & & & \quad x \text{ a message} \,\} \end{array}$$

is a bisimulation wrt. the set L_1, where $\hat{s}$ in L are obtained from the lists s in K by first eliminating all garbled messages and then clearing all remaining messages of their c tags. For a list $s = a_c a_g a_c b_c b_c c_g a_c$, for instance, $\hat{s} = aabba$. Proving that $\mathcal{F}$ is a bisimulation is not difficult, and most of the involvement of the user goes into theorems describing, for instance, the relation between $\hat{s}$ and $\hat{s}'$ if s' is obtained from s by losing one message. Provided with these theorems, Isabelle proves by one single application of `auto_tac` that $\mathcal{F}$ is a bisimulation, automatically guessing, for instance, the matching weak transitions. The proof script contains less than 300 lines, and has been set up within a few hours only.

Remark: We have considered a faulty filter F which loses and duplicates messages itself. For a correct filter F' we do not have $K \parallel F' \approx_{L_1} L$. Intuitively, if the filter is correct then $K \parallel F'$ is more reliable than L, while observational equivalence requires the two systems to have the same (un)reliability. In this case one can use *observational preorder*, in which L has to simulate the behaviour of $K \parallel F'$ but not vice versa.

3.2 The Alternating Bit Protocol

The Alternating Bit Protocol (ABP), proposed in [5, 1], is a well-established benchmark for proof methodologies implemented in theorem provers (see, for instance, [10, 2, 9, 3]). It turns unreliable channels into reliable communication lines. We consider an infinite-state variant in which the channels can hold arbitrarily many messages, like in the previous example. We assume that messages can be lost and duplicated, but not garbled. In order to construct a protocol also dealing with (detectable) garbling we can use the result

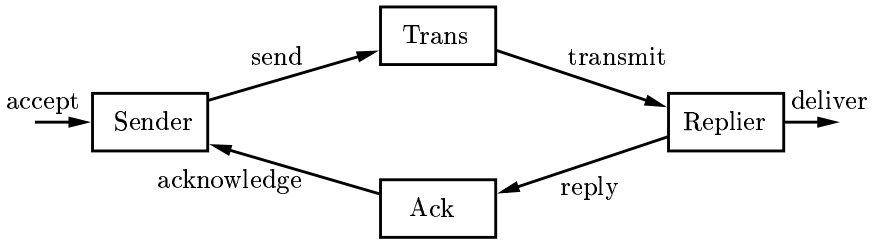

Figure 1: Components of the ABP. The protocol is designed to turn the unreliable channels `Trans` and `Ack` into a reliable communication line.

System: sender and receiver

S1	$S_a(b) \xrightarrow{\text{acc}} S_s(b)$	**R1**	$R_a(b) \xrightarrow{\overline{\text{del}}} R_s(b)$	(accept/deliver)
S2	$S_s(b) \xrightarrow{\overline{\text{cs}}(b)} S_w(b)$	**R2**	$R_s(b) \xrightarrow{\overline{\text{cr}}(b)} R_w$	(transmit)
S3	$S_w(b) \xrightarrow{\tau} S_s(b)$	**R3**	$R_w(b) \xrightarrow{\tau} R_s(b)$	(timeout)
S4	$S_w(b) \xrightarrow{\text{ca}(b)} S_a(\neg b)$	**R4**	$R_w(b) \xrightarrow{\text{ct}(\neg b)} R_a(\neg b)$	(receive new copy)
S5	$S_w(b) \xrightarrow{\text{ca}(\neg b)} S_w(b)$	**R5**	$R_w(b) \xrightarrow{\text{ct}(b)} R_w(b)$	(receive old copy)

Specification: one-place buffer

B1	$B_a \xrightarrow{\text{acc}} B_d$	**B2**	$B_d \xrightarrow{\overline{\text{del}}} B_a$	(accept/deliver)

Table 2: Implementation and Specification of the ABP. Sender S (**S1**–**S5**) and Replier R (**R1**–**R5**) repeatedly transmit copies of the current message (acknowledgement) until they receive an acknowledgement (a new message). Old copies are distinguished from new ones with the help of an alternating bit b. The ABP is supposed to turn faulty channels of arbitrary size into reliable one-place buffers B (**B1**–**B2**). The indices refer to the states S, R, and B can assume.

of the previous subsection, replacing a channel by the parallel composition of a channel and a filter. The compositionality of observational equivalence automaticall guarantees the correctness of this new protocol.

The model as well as the outline of the proof follow [6]. The protocol is modelled as the parallel composition of four processes: two unreliable channels—one channel, `Trans` or T for short, *transmitting* messages, and the other, `Ack` or A for short, returning *acknowledgements*—, a *sender* S, and a *replier* module R; see Figure 1 for a schematic view.

The channels A and T are formalised exactly as channel L in the previous subsection, and so we omit this part. The behaviour of sender and replier, as well as of the specification, is formally described in Table 2. The initial

states of sender and receiver are $S_a(0)$ and $R_s(0)$, respectively. The *sender* module continuously accepts messages from the environment (**S1**), transmits them repeatedly over channel T (**S2**, **S3**) and waits for the current acknowledgement along A (**S4**, **S5**), before accepting a new message. After having delivered a message to the environment (**R1**), the replier R repeatedly transmits tagged acknowledgements to the sender (**R2**, **R3**) until a new message arrives (**R4**, **R5**). The initial state of the protocol is the process

$$ABP = S_a(0) \parallel T(\epsilon) \parallel A(\epsilon) \parallel R_s(1)\,.$$

The system should behave like a buffer of capacity one. This specification is also formalised in Table 2, the initial state being B_a. We show that $ABP \approx_{L_1} B_a$. Recall form the previous section that $L_1 = \{\mathrm{acc}, \overline{\mathrm{del}}\}$.

The states of the protocol can be divided in two classes: a message can either be accepted or delivered, possibly after a finite number of silent transitions. The bisimulation relation $\mathcal{B}$ thus falls into the two corresponding sub-relations

$$\begin{array}{rcll}
\mathcal{B}_a & \stackrel{\text{def}}{=} & \{ & (S_a(\neg b) \parallel T(b^n) \parallel A(b^p) \parallel R_s(b), & B_a) \\
& & & (S_a(\neg b) \parallel T(b^n) \parallel A(b^p) \parallel R_w(b), & B_a) \\
& & & (S_s(\neg b) \parallel T((\neg b)^n) \parallel A(b^p(\neg b)^q) \parallel R_s(\neg b), & B_a) \\
& & & (S_s(\neg b) \parallel T((\neg b)^n) \parallel A(b^p(\neg b)^q) \parallel R_w(\neg b), & B_a) \\
& & & (S_w(\neg b) \parallel T((\neg b)^n) \parallel A(b^p(\neg b)^q) \parallel R_s(\neg b), & B_a) \\
& & & (S_w(\neg b) \parallel T((\neg b)^n) \parallel A(b^p(\neg b)^q) \parallel R_w(\neg b), & B_a) \mid b \in \{0,1\} \}
\end{array}$$

capturing those states eventually accepting a new message, and

$$\begin{array}{rcll}
\mathcal{B}_d & \stackrel{\text{def}}{=} & \{ & (S_s(\neg b) \parallel T((\neg b)^m) \parallel A(b^p) \parallel R_a(\neg b), & B_d) \\
& & & (S_w(\neg b) \parallel T((\neg b)^m) \parallel A(b^p) \parallel R_a(\neg b), & B_d) \\
& & & (S_s(\neg b) \parallel T((\neg b)^m b^n) \parallel A(b^p) \parallel R_s(\neg b), & B_d) \\
& & & (S_s(\neg b) \parallel T((\neg b)^m b^n) \parallel A(b^p) \parallel R_w(\neg b), & B_d) \\
& & & (S_w(\neg b) \parallel T((\neg b)^m b^n) \parallel A(b^p) \parallel R_s(\neg b), & B_d) \\
& & & (S_w(\neg b) \parallel T((\neg b)^m b^n) \parallel A(b^p) \parallel R_w(\neg b), & B_d) \mid b \in \{0,1\} \}
\end{array}$$

where eventually a message will be delivered. In every channel, there are at most two types of messages or acknowledgements: copies of the one that currently has to be delivered, and possibly copies of the previous one. The finite lists in T and A are thus either of the form x^n, or $x^n y^m$; in Isabelle this can be expressed in terms of the `replicate` operator from the built-in theory for finite lists.

To show that $\mathcal{B} \stackrel{\text{def}}{=} \mathcal{B}_a \cup \mathcal{B}_d$ is indeed a bisimulation wrt. $\{\,\text{acc}, \overline{\text{del}}\,\}$, we follow our usual scheme. As a typical example, consider the case where the ABP performs a strong acc transition. We have to prove the obligation, if $(P, Q) \in \mathcal{B}$ and $P \xrightarrow{\text{acc}} P'$, then there exists a state Q' such that $Q \stackrel{\text{acc}}{\Longrightarrow} Q'$, and $(P', Q') \in \mathcal{B}$. Out of the six subrelations in $\mathcal{B}_a$ differing in the shape of P, Isabelle automatically extracts the first two as those in which P can do an acc. It remains to show that in both cases the resulting process P' fits the shape of the left process in the third and fourth subrelations of $\mathcal{B}_d$. The difficulty of this proof step results from the lists in T and A in $\mathcal{B}_d$ looking differently from those in $\mathcal{B}_a$. Once provided with the necessary theorems about finite lists, however, Isabelle completes it fully automatically.

Another interesting example is the reverse case, in which $Q \stackrel{\text{acc}}{\Longrightarrow} Q'$ and $P \stackrel{\text{acc}}{\Longrightarrow} P'$. For the third through sixth case of $\mathcal{B}_a$, the user has to provide suitable sequences of weak transitions leading to the acceptance of a new message. In all of the cases, we can apply the following scheme: remove all messages and acknowledgements from T and A (that this is possible can be shown once in a separate proof, by an induction on the length of the lists stored in the channels), then have R transmit an acknowledgement to S, and finally execute the acc transition.

The proof script contains about 800 lines. As nearly half of it consists of theorems about the finite lists used in the channels, some experience with theorem provers is necessary to set up the proofs. The bisimulation part itself can be set up within a few days by a user experienced both in the bisimulation proof method and theorem proving. In particular, only a few proof procedures strongly based on Isabelle's automatic tactics are necessary to capture all of the almost 100 proof obligations. As pointed out in [6], this example is clearly on the edge of what can be proved without machine assistance, if not beyond.

4 Using Compositionality

Various techniques have been developed with the intention to reduce the size of bisimulation proofs. The basic idea behind all these techniques is to reuse results that have already been proved before. In this section, we focus on replacing larger sub-components with their specifications; in the next section, we present a way of obtaining bisimilarity results by examining subsets of bisimulations instead of entire bisimulations.

A major characteristic of observation equivalence is that it is *compositional* [6]. This means that in bisimulation proofs about large composite systems, it is possible to replace sub-systems with their specifications, and

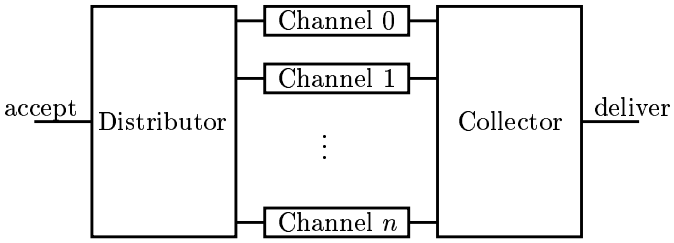

Figure 2: A Specification of the SWP. The parameterized protocol consists of a parallel composition of n communication lines implementing the ABP.

System: distributor and collector

D1	$D_n^h \xrightarrow{\mathrm{acc}(x)} D_n^h(x)$	**C1**	$C_n^l \xrightarrow{\mathrm{co}(x)} C_n^{l+_n 1}(x)$	(accept/collect)
D2	$D_n^h(x) \xrightarrow{\overline{\mathrm{ci}}_h(x)} D_n^{h+_n 1}$	**C2**	$C_n^l(x) \xrightarrow{\overline{\mathrm{del}}(x)} C_n^l$	(distribute/deliver)

Specification: buffer of capacity $n+2$

B3	$B^{n+2}(s) \xrightarrow{\mathrm{ci}(x)} B^{n+2}(sx)$	if $\lvert s\rvert < n+2$	(accept)
B4	$B^{n+2}(xs) \xrightarrow{\mathrm{co}(x)} B^{n+2}(s)$		(deliver)

Table 3: Distributor and collector of the SWP. An SWP with n channels can be specified by an $n+2$-buffer.

continue the proof with the resulting system, which is usually considerably smaller. Further, with observation equivalence preserving safety properties, this technique is also applicable in order to reduce the size of a system before verifying safety properties. This is especially interesting if observation equivalence reduces an infinite system to a finite specification; in a combination of theorem proving and model checking, the infinite system can be reduced interactively to its finite specification in a first step, and in a second step, safety properties can be checked fully automatically upon the specification.

4.1 A Specification of the Sliding Window Protocol

In [10], a simple parameterized Sliding Window Protocol (SWP) with input and output windows of equal size is presented. The system consists of n communication lines each of which uses the ABP on faulty channels. Figure 2 shows a schematical view. Incoming messages are cyclically distributed to the communication lines by a *distributor* D_n, and are recollected and delivered by a *collector* C_n; their behaviour is formally described in Table 3 (**D1**, **D2**,

B5	$B^A(s) \xrightarrow{\mathrm{ci}_h(x)} B^A(s[h := x])$	if $s!h = \Box$	(accept)
B6	$B^A(s) \xrightarrow{\overline{\mathrm{co}}_h(x)} B^A(s[h := \Box])$	if $s!h = x \neq \Box$	(deliver)

Table 4: An *array buffer* B^A containing lists of length n.

and **C1**, **C2**). The initial state of the SWP is the process

$$SWP \quad = \quad D_n^h \parallel ABP_1 \parallel \ldots \parallel ABP_n \parallel C_n^l$$

where ABP_i is a copy of the process of the last section[2]. The system should behave like a buffer B^{n+2} of capacity $n+2$, with n being the number parallel channels in the system (the capacity is $n+2$ and not n because the distributor and the collector contribute with one place each), see also Table 3 (**B1**, **B2**). This time we cannot abstract from data, as the system needs not deliver a message before accepting a new one: we have to guarantee that messages not be swapped. So we prove $AWP \approx_{L_2} B^{n+2}$ where $L_2 = \{\, \mathrm{acc}(x), \overline{\mathrm{del}}(x) \mid m \in M \,\}$.

The proof falls into several parts, and makes extensive use of compositionality.

(1) In a first step, the ABP components are replaced with one-place buffers, using the result from the previous section.

$$(D_n \parallel ABP_1 \parallel \ldots \parallel ABP_n \parallel C_n) \approx_{L_1} (D_n \parallel B_{a1} \parallel \ldots \parallel B_{an} \parallel C_n)$$

Observe that we have got rid of the infinite-state processes, but we still have to carry out a parametric proof, valid for all n.

(2) We need a finite representation of the n one-place buffers put in parallel. They can be described by a single component B^A, which we refer to as an *array buffer*. For every buffer in the SWP currently storing a message x, the corresponding cell of B^A contains x as well; if the buffer is empty, B^A contains an auxiliary element $\Box$; see Table 4 for a formal description. We use $\Box$ to denote empty cells. We show that $B^A(\Box^n) \approx_{L2} (B_{a1} \parallel \ldots \parallel B_{an})$, where $L_2 = \{\, \mathrm{ci}_h(v), \overline{\mathrm{co}}_h \mid h \in I\!N \,\}$, using induction and the bisimulation proof method described in Section 3, and so, exploiting compositionality,

$$(D_n \parallel B_{a1} \parallel \ldots \parallel B_{an} \parallel C_n) \approx_{L_1} (D_n \parallel B^A(\Box^n) \parallel C_n)$$

(3) We proceed by comparing the resulting system to another system given by a parallel version of an n-place buffer, B^P (see **B7** and **B8** in

[2]the actions of these processes need to be suitably renamed.

B7	$B^P(h,l,s) \xrightarrow{\mathrm{ci}(x)} B^P(h +_{\lvert s\rvert} 1, l, s[h := x])$	if $s!h = \Box$	(accept)
B8	$B^P(h,l,s) \xrightarrow{\overline{\mathrm{co}}(x)} B^P(h, l +_{\lvert s\rvert} 1, s[l := \Box])$	if $s!l = x \neq \Box$	(deliver)

Table 5: The parallel buffer B^P.

Table 5), and two barrier one-place buffers, one attached to its front, and another to its back. Internally, B^P is organized like B^A, yet it possesses only one input and one output connection, and stores and retrieves messages in a cyclic order. We show

$$(D_n \parallel B^A(\Box^n) \parallel C_n) \approx_{L_1} (B_a \parallel B^P(\Box^n) \parallel B_a)$$

again by exhibiting a bisimulation.

(4) To complete the proof we have to show by exhibiting suitable bisimulations that B^P with its two one-place buffers behaves like the $n+1$-buffer from the specification:

$$(B_a \parallel B^P(\Box^n) \parallel B_a) \approx_{L_1} B^{n+2}(\Box^n)$$

The proof script with the bisimulations verifying the SWP contains about 600 lines, and has been set up in less than two weeks. The proofs of (2) and (3) are rather straightforward, as the processes related by the bisimulations behave in similar ways. Isabelle therefore deduces the proofs automatically without the user having to split them into single theorems covering the obligations. Note that the weak transitions are directly derivable from strong ones, thus need not be given by the user. Also, almost no additional results about the lists have to be provided by the user. The most challenging part concerning the mechanization was the proof of the first part of (4), as here the bisimulation maps the cyclic lists of B^P to the linear lists stored in $B^{n+2}(\Box^n)$. The corresponding theorems make up for nearly two thirds of the proof; their derivation necessitates certain expertise in theorem proving. For more information about the above case-studies, see [16].

5 Reducing Proofs with 'Up to'-Techniques

In theorem proving, one is usually interested in splitting large proofs into smaller sub-proofs in order to gain maximal profit from the automatic tactics. In Section 3, we have achieved this by proving bisimulation obligations in separate theorems. In Section 4, we have exploited compositionality of observation equivalence. Here, we discuss the application of 'up to'-techniques

that have been proposed by Sangiorgi and Milner in [19] in order to reduce the size of the relations. That is, instead of showing that a relation $\mathcal{R}$ is a bisimulation, one chooses some suitable $\mathcal{S}$ and proves that $\mathcal{S} \circ \mathcal{R} \circ \mathcal{S}^{-1}$ is a bisimulation. (Observe that the size of $\mathcal{S} \circ \mathcal{R} \circ \mathcal{S}^{-1}$. can be far larger than the size of $\mathcal{S}$ and $\mathcal{R}$.) The theory of 'up to'-techniques—in particular, how a suitable $\mathcal{S}$ should look like—has been further investigated by Sangiorgi [18]. In this section, we are concerned with the application in theorem proving of one such technique, called 'up to expansion'.

Intuitively, an expansion is a bisimulation in which the first process performs at least as many internal steps as, or is less efficient than, the second one. Expansion can be used to abstract, for instance, from internal timers, or from internal communication of data. Every expansion (wrt. L) is a bisimulation (wrt. L). The converse does not hold.

Definition 2 (Expansion) *$\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is an* expansion wrt. *$L \subseteq \mathcal{A}ct$ if for all $P\mathcal{R}Q$, all $\alpha \in L \cup \{\tau\}$, all $P' \in \mathcal{S}_1$, and all $Q' \in \mathcal{S}_2$,*

(i) If $P \xrightarrow{\alpha} P'$, there exists $Q' \in \mathcal{S}_2$ such that $Q \xrightarrow{\hat{\alpha}} Q'$ and $P'\mathcal{R}Q'$.

(ii) If $Q \xrightarrow{\alpha} Q'$, there exists $P' \in \mathcal{S}_1$ such that $P \xRightarrow{\alpha} P'$ and $P'\mathcal{R}Q'$.

We call $\geq_L$ the union of all expansions wrt. L.

Definition 3 (Bisimulation up to Expansion) *$\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is a* bisimulation up to expansion wrt. *$L \subseteq \mathcal{A}ct$ if for all $P\mathcal{R}Q$, all $\alpha \in L \cup \{\tau\}$, all $P' \in \mathcal{S}_1$, and all $Q' \in \mathcal{S}_2$,*

(i) If $P \xrightarrow{\alpha} P'$, there exist $P'' \in \mathcal{S}_1$ and $Q', Q'' \in \mathcal{S}_2$ such that $Q \xRightarrow{\hat{\alpha}} Q'$ and $P' \geq_L P''\mathcal{R}Q'' \leq_L Q'$.

(ii) If $Q \xrightarrow{\alpha} Q'$, there exists $P', P'' \in \mathcal{S}_1$ and $Q'' \in \mathcal{S}_2$ such that $P \xRightarrow{\hat{\alpha}} P'$ and $P' \geq_L P''\mathcal{R}Q'' \leq_L Q'$.

It is a standard exercise to show that two states are observationally equivalent (wrt. L) if they are bisimilar up to expansion (wrt. L).

5.1 Write-Invalidate Cache Coherence

As a last case study, we have chosen an example which is parameterized both in number and size of its components. An interesting point is that internal communications follow a broadcast mechanism. In our formalism this can be easily taken care of by modifying the introduction rules for parallel composition. The new rules are

The memory, used by specification and system

M1	$M(s) \xrightarrow{\mathrm{rq}_r(i)} M(s,i)$		(lookup)
M2	$M(s,i) \xrightarrow{\overline{\mathrm{ret}}(s[i])} M(s)$		(return)
M3	$M(s) \xrightarrow{\mathrm{rq}_w(i,v)} M(s\{v/s[i]\})$		(update)

Specification: interfaces to memory without caches

I1	$I_l(\mathrm{stat},\mathrm{free}) \xrightarrow{\mathrm{read}_l(i)} I_l(\mathrm{stat},\mathrm{read}(i))$	if $i \leq \lvert s\rvert$	(read)
I2	$I_l(\mathrm{awake},\mathrm{read}(i)) \xrightarrow{\overline{\mathrm{rq}}_r(i)} I_l(\mathrm{awake},\mathrm{wait})$		(lookup)
I3	$I_l(\mathrm{awake},\mathrm{wait}) \xrightarrow{\mathrm{ret}(v)} I_l(\mathrm{awake},\mathrm{del}(v))$		(return)
I4	$I_l(\mathrm{stat},\mathrm{del}(v)) \xrightarrow{\overline{\mathrm{return}}_l(v)} I_l(\mathrm{stat},\mathrm{free})$		(serve)
I5	$I_l(\mathrm{stat},\mathrm{free}) \xrightarrow{\mathrm{write}_l(i,v)} I_l(\mathrm{stat},\mathrm{write}(i,v))$	if $i \leq \lvert s\rvert$	(write)
I6	$I_l(\mathrm{awake},\mathrm{write}(i,v)) \xrightarrow{\overline{\mathrm{rq}}_w(i,v)} I_l(\mathrm{awake},\mathrm{free})$		(update)
I7	$I_l(\mathrm{awake},\mathrm{req}_l) \xrightarrow{\mathrm{rq}_r(i)} I_l(\mathrm{asleep},\mathrm{req}_l)$		(sleep)
I8	$I_l(\mathrm{asleep},\mathrm{req}_l) \xrightarrow{\mathrm{ret}(v)} I_l(\mathrm{awake},\mathrm{req}_l)$		(wake up)
I9	$I_l(\mathrm{awake},\mathrm{req}_l) \xrightarrow{\mathrm{rq}_w(i,v)} I_l(\mathrm{awake},\mathrm{req}_l)$		(ignore)

System: interface to memory with caches

C2a	$I_l^c(c_l,\mathrm{stat},\mathrm{read}(i)) \xrightarrow{\tau} I_l^c(c_l,\mathrm{stat},\mathrm{wait})$	if $\mathrm{valid}(c_l[i])$	(cache)
C2b	$I_l^c(c_l,\mathrm{awake},\mathrm{read}(i)) \xrightarrow{\overline{\mathrm{rq}}_r(i)} I_l^c(c_l,\mathrm{awake},\mathrm{wait})$	if $\mathrm{invalid}(c_l[i])$	(lookup)
C6	$I_l^c(c_l,\mathrm{awake},\mathrm{write}(i,v)) \xrightarrow{\overline{\mathrm{rq}}_w(i,v)} I_l^c(c_l\{(v,\mathrm{valid})/i\},\mathrm{awake},\mathrm{free})$		(update)
C9	$I_l^c(c_l,\mathrm{awake},\mathrm{req}_l) \xrightarrow{\mathrm{rq}_w(i,v)} I_l^c(c_l\{(c_l[i],\mathrm{invalid})/i\},\mathrm{awake},\mathrm{req}_l)$		(invalidate)

Table 6: Write invalidate cache coherence. Rules **M1** to **M3** describe the behaviour of the memory. Rules **I1** through **I9** specify the behaviour of each interface. The behaviour of the implementation is given by rules **C1** through **C9** similar to **I1** to **I9** except for the rules that are explicitely shown above, which involve cache cells. Rules **I1**–**I4**, respectively **C1**–**C4**, describe the behaviour of interfaces I_l and I_l^c in the presence of a read request. **I5**, **I6** and **C5**, **C6** handle write requests. **I7**, **I8** and **C7**, **C8** describe how the interfaces fall asleep whenever they encounter a read request on the bus. **I9** and **C9** concern the detection that some other interface writes on the memory.

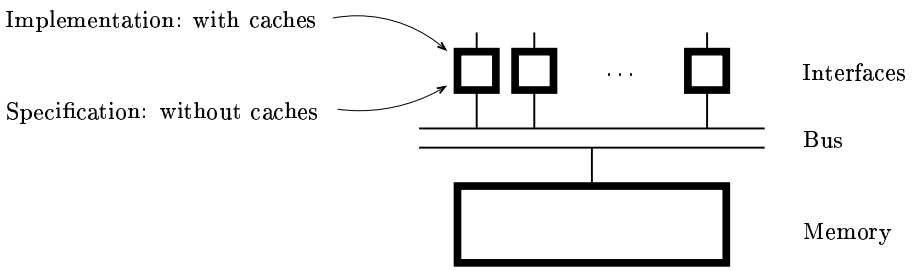

Figure 3: The structure of implementation and specification. In order to read from or write to the memory, processors have to connect to one of the interfaces. The interfaces of the implementation possess caches, whereas those of the specification have to pass every read request to the main memory.

- $P_i \xrightarrow{\alpha} P_i'$ implies $P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_n \xrightarrow{\alpha} P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_n$;
- $P_i \xrightarrow{\bar{a}(v)} P_i'$ and $\forall j \neq i.\ P_j \xrightarrow{a(v)} P_j'$ imply $P_1 \parallel \ldots \parallel P_n \xrightarrow{\tau} P_1' \parallel \ldots \parallel P_n'$.

The system: implementation and specification We consider a set of *interfaces* connected to a central *memory* M by a *bus system*, see Figure 3 for a graphical overview. In order to access M (Table 6, **M1**–**M3**), a *processor* has to connect to one of the interfaces which then serves its read or write request. In the implementation, each interface I_i^c (Table 6, **C1**–**C9**) contains a cache of its own, so that a part of the read requests can be served without accessing the bus. We consider a simple cache protocol, the informal description of which is based on [4], and which has already been proved correct in a formal environment [7]. In the specification, the interfaces I_i (Table 6, **I1**–**I9**) do not possess caches, but pass every read request to the memory M along the bus.

In case of a read request, an interface I_i^c checks if the corresponding cache cell is valid, otherwise it fetches the value from the memory (**C1**–**C4**); its implementation I_i always looks up the memory (**I1**–**I4**). Write requests are served by updating the memory as well as the concerned cache cell (**C5**–**C6** and **I5**–**I6**). By continually snooping on the bus, interfaces notice read and write requests sent along it. They wait upon a read request, until it has been served (**C7**–**C8** and **I7**–**I8**), before accessing the bus themselves. Write requests are ignored, only that interfaces I_l^c invalidate the concerned cache cell (**C9** and **I9**). Note that Table 6 only displays those axioms among **C1**–**C9** that considerably differ from **I1**–**I9**, because some cache is involved.

Cache cells can either be *valid* or *invalid*. We show that the system with the caches is observationally equivalent to the system without the caches. As

a consequence, the cache system can be safely applied instead of a system without caches. Moreover, this result holds for systems of arbitrary size.

Let L_3,

$$\begin{array}{rcl} L_3 & \stackrel{\text{def}}{=} & \{\, \mathrm{read}_l(i),\ \mathrm{write}_l(i,v),\ \mathrm{return}_l(v) \mid l, i \in I\!N,\ v \text{ of suitable type} \,\} \\ & \cup & \{\, \overline{\mathrm{read}}_l(i),\ \overline{\mathrm{write}}_l(i,v),\ \overline{\mathrm{return}}_l(v) \mid l, i \in I\!N,\ v \text{ of suitable type} \,\} \end{array}$$

contain the visible behaviour of the systems consisting of communications between processors and interfaces. Then,

$$\mathrm{Mem}(s) \parallel I_1^c(t_1) \parallel \cdots \parallel I_n^c(t_n) \approx_{L_3} \mathrm{Mem}(s) \parallel I_1(u_1) \parallel \cdots \parallel I_n(u_n)$$

for every content s of the memory, and for all possible tuples of parameters $t_1, \ldots, t_n$ and $u_1, \ldots u_n$, where $t_l = (c_l, \mathrm{stat}_l, \mathrm{req}_l)$ and $u_l = (\mathrm{stat}_l, \mathrm{req}_l)$, and $\mathrm{valid}(c_l[i])$ implies $c_l[i] = s[i]$, for all i. Note that this last condition reflects the safety property of the protocol, which is guaranteed by observation equivalence.

Remark: Note that a value returned to the environment as the value of some cell by an interface need not necessarily coincide with the current value of that cell, neither in the implementation nor in the specification. This is because an interface may delay the delivery of a value it has already retrieved from the memory or from its cache, even until the cell has been updated in the memory by another interface.

Bisimulation up to expansion For fixed but arbitrary n and m, the (parameterized) relation $\mathcal{R}_{n,m}$ contains pairs of states of the implementation and the specification in which all the interfaces are awake, that is, are allowed to access the bus immediately, all caches are coherent, and in which the requests in the I_l^c and I_l are identical for all l.

$$\begin{array}{rcl} \mathcal{R}_{n,m} & \stackrel{\text{def}}{=} & \{\ (\mathrm{Mem}(s) \parallel I_1^c(c_1, \mathrm{awake}, \mathrm{req}_1) \parallel \ldots \parallel I_n^c(c_n, \mathrm{awake}, \mathrm{req}_n), \\ & & \quad \mathrm{Mem}(s) \parallel I_1(\mathrm{awake}, \mathrm{req}_1) \quad \parallel \ldots \parallel I_n(\mathrm{awake}, \mathrm{req}_n)) \quad \mid \\ & & \quad |s| = m \ \wedge\ \forall\, 1 \le l \le n,\ 1 \le i \le m.\ \ \mathrm{valid}(c_l[i]) \ \Rightarrow\ c_l[i] = s[i] \ \} \end{array}$$

The relation does not consider states in which interfaces are asleep, that is, where one of the interfaces has sent a read request on the bus. These are dealt with by expansion relations $\mathcal{E}^c_{n,m}$ and $\mathcal{E}_{n,m}$; for details refer to [15]. They relate states of the implementation, respectively of the specification, directly before and after a read request has been served by the main memory. These states are observationally indistinguishable.

The proof falls into the following steps: (1) prove that identity on states, $\mathcal{I}d$, every $\mathcal{E}^c_{n,m} \cup \mathcal{I}d$, and every $\mathcal{E}_{n,m} \cup \mathcal{I}d$ are expansions wrt. L_3; and, (2) show that every $\mathcal{R}_{n,m}$ is a bisimulation up to expansion wrt. L_3, exploiting the results proved in step (1). The proof follows the same pattern as those in Section 3, see [15] for details.

The Isabelle/HOL proof script consists of approximately 1000 lines of code. We were able to increase the profit from Isabelle's automatic tactics by splitting the proof obligations into separate subgoals. We further benefitted from the uniform structure of the single proofs, so we were able to use the same proof script for several obligations.

Standard bisimulation In a standard bisimulation proof, a lot more combinations have to be considered; both systems can be asleep, for instance, or one system is asleep waiting for a read request to be served while the other is awake. This yields bisimulation relations $\mathcal{R}^{st}_{n,m} \stackrel{\text{def}}{=} \mathcal{R}_{n,m} \cup \mathcal{R}^{ws}_{n,m} \cup \mathcal{R}^{ss}_{n,m}$ wrt. L_3, where $\mathcal{R}^{ws}_{n,m}$ contains those pairs of states in which the implementation is awake and the specification is asleep, and $\mathcal{R}^{ss}_{n,m}$ contains those in which both are asleep. Pairs in $\mathcal{R}^{ws}_{n,m}$ are due to an I^c_l retrieving a value from its cache while the corresponding I_l has to look it up in the main memory. The combination $\mathcal{R}^{sw}_{n,m}$, in which the interfaces of the implementation are asleep while those of the specification are not, need not be considered: whenever an I^c_l sends a read request along the bus, there is no need for the corresponding I_l to go a step further and serve the request.

In order to reduce the number of cases of this proof, we have modified the transition systems so that interfaces that are asleep cannot even communicate with the environment. But still the proof script is as long as that using bisimulations up to expansion, that is, around 1000 lines. Without the simplification, it would have been even longer.

6 Discussion

In the previous sections, we have described how to obtain a general verification technique for systems with an infinite number of states, by applying the proof method of observational equivalence in interactive theorem-proving, and have examined it on various benchmark examples. We now present conclusions drawn from these case-studies, discussing both advantages and deficiencies of the approach.

Range of applicability Observational equivalence is a natural specification technique for communication protocols, in which the specification can be described as an abstract system (for instance, a buffer of some type), which must be implemented in a distributed way, or with components subject to failure. It is less natural for specifications consisting of a list of properties the system should satisfy, such as distributed algorithms[3] The technique is most suitable for systems that can be concisely described but have infinitely many states and use some nontrivial datatypes. These systems are still out of reach for fully automatic tools, but lead to manageable bisimulations.

Is observational equivalence suitable? Observational equivalence has been argued to be too discriminating in practice; in fact, often language (or, trace) equivalence [6] is preferred. In the area of communication protocols, this question seems to be a lesser problem. Many specifications are deterministic, and in this observation and fair testing equivalence [8]—and sometimes even language equivalence—coincide. Compared to these two equivalences, observational equivalences offers a better proof methodology. Thus, in cases where the equivalences coincide, one can profit from bisimulation proof techniques in order to show language or testing equivalence.

Sometimes one might be interested in over-specifications. In these cases, *observational preorder* offers proof techniques based on *simulations*, considering only one direction of a bisimulation. The resulting proof methodology is similar, and compositionality can be exploited as well.

Keeping bisimulations manageable Keeping the size of relations manageable is an important problem of our approach. The compositionality of observation equivalence is a big help, as we could see in Section 4: if we had not been able to replace the ABP channels by one-place buffers, the bisimulation would have been unmanageable. Furthermore, there exist various up-to techniques, one of which is bisimulation up to expansion, see Section 5.

How to find a bisimulation? Searching for a bisimulation is an incremental process. Usually, one starts with some base state of implementation and specification, and adds pairs, or (probably infinite) families of pairs, until one has obtained a bisimulation. This approach can be formally described in terms of the *coinductive* method of fixed-point generation (see, for instance,

[3]For some distributed algorithms one can consider the most liberal system satisfying the properties, and prove that the implementation behaves "as well, or better" than this system. For this purpose, observational equivalence is replaced by the observational preorder.

[17]), and is supported by Isabelle/HOL [12]. The advantage of coinduction is that finding the relation and proving that it is a bisimulation are intertwined, and Isabelle deals with all technical details as, for instance, "has pair (s, t) already been considered or not?" One inconvenience is, however, that the bisimulation cannot yet be extracted as an Isabelle constant from the coinductive proof, to be available for further use.

Dealing with data structures Proving simple facts about the data structures of a system (list, stacks, et cetera) may amount to more than half of the interaction with the theorem prover. These facts are stored in Isabelle's database for future use, but their application still requires considerable expertise in theorem proving. The user may decide not to perform the full proof, by taking theorems about data structures as unproved axioms. For simple theorems this is a sensible approach, since the proof loses almost no credibility.

General evaluation and future work The approach is certainly labour intensive when compared to automatic verification. It is useful for not too large systems with an infinite state space which do not exhibit regularity properties making it amenable to model checking. The approach is particularly suitable for modular systems in which each of the modules has a separate specification. Future work should concentrate in the interactive design of the bisimulation. As mentioned above, coinduction techniques for this problem are available in Isabelle, but they are still very unfriendly to the user.

References

[1] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Comm. of the ACM*, 12(5):260–261, May 1969.

[2] M. Bezem and J. F. Groote. A formal verification of the alternating bit protocol in the calculus of constructions. Logic Group Preprint Series 88, Dept. of Philosophy, Utrecht University, 1993.

[3] E. Gimenez. An application of co-inductive types in Coq: Verification of the alternating bit protocol. In *Proc. TYPES'95*, volume 1158 of *LNCS*, pages 135–152. Springer, 1996.

[4] J. Hennessy and D. Paterson. *Computer Architecture – A Quantitative Approach.* Morgan Kaufmann, 1996.

[5] W. C. Lynch. Reliable full-duplex file transmission over half-duplex telephone lines. *Comm. of the ACM*, 11(6):407–410, 1968.

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

[7] J. S. Moore. An ACL2 proof of write invalidate cache coherence. In *Proc. CAV'98*, volume 1427 of *LNCS*, pages 29–38. Springer, 1998.

[8] V. Natarajan and R. Cleaveland. Divergence and fair testing. In *Proc. ICALP'95*, volume 944 of *LNCS*, pages 648–659. Springer, 1995.

[9] T. Nipkow and K. Slind. I/O automata in Isabelle/HOL. In *Proc. TYPES'94*, volume 996 of *LNCS*, pages 101–119. Springer, 1994.

[10] K. Paliwoda and J. Sanders. The sliding-window protocol. Technical Report PRG-66, Programming Research Group, Oxford University, March 1988.

[11] D. M. R. Park. *Concurrency and Automata on Infinite Sequences*, volume 104 of *LNCS*. Springer, 1980.

[12] L. C. Paulson. Isabelle's object-logics. Technical Report 286, University of Cambridge, Computer Laboratory, 1993.

[13] L. C. Paulson. *Isabelle: a generic theorem prover*, volume 828 of *LNCS*. Springer, 1994.

[14] G. Plotkin. Structural operational semantics. Technical report, DAIMI, Aarhus University, 1981.

[15] C. Röckl. Proving write invalidate cache coherence with bisimulations in Isabelle/HOL. In *Proc. of FBT'00*, pages 69–78. Shaker, 2000.

[16] C. Röckl and J. Esparza. Proof-checking protocols using bisimulations. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*, pages 525–540. Springer, 1999.

[17] J. Rutten. Automata and coinduction (an exercise in coalgebra). In *Proc. CONCUR'98*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.

[18] D. Sangiorgi. On the bisimulation proof method. *Math. Struct. in Comp. Science*, 1998. A summary appeared in Proc. MFCS'95.

[19] D. Sangiorgi and R. Milner. The problem of weak bisimulation up-to. In *Proc. CONCUR'92*, volume 630 of *LNCS*, pages 32–46. Springer, 1992. A revised version has appeared as a technical report.

[20] J. L. A. Snepscheut. The sliding-window protocol revisited. *Formal Aspects of Computing*, 7:3–17, 1995.