The Computer Journal Advance Access published July 21, 2006

doi:10.1093/comjnl/bxl039

Verifying an ATM Protocol Using a Combination of Formal Techniques*

VLAD RUSU

IRISA/INRIA Rennes, Campus de Beaulieu, 35042 Rennes Cedex, France
E-mail: rusu@irisa.fr

This paper describes a methodology and a case study in formal verification. The case study is the SSCOP protocol, a member of the ATM adaptation layer whose main role is to perform a reliable data transfer over an unreliable communication medium. The methodology involves (i) simulation for initial debugging; (ii) partial-order abstraction that preserves the properties of interest and (iii) compositional verification of the properties at the abstract level using the PVS theorem prover. Steps (ii) and (iii) guarantee that the properties still hold on the whole (composed, concrete) system. The value of the approach lies in adapting and integrating several existing formal techniques into a new verification methodology that is able to deal with real case studies.

Keywords: Compositional, deductive, partial-order based verification

Received 16 August 2005; revised 29 May 2006

1. INTRODUCTION

Formal verification methods form a wide spectrum ranging from fully automatic *model checking* to purely interactive *theorem proving*. Yet, despite many success stories, it appears that most real, industrial systems are out of reach of current state-of-the art model checkers [1, 2, 3, 4] and theorem provers [5, 6, 7, 8].

Indeed, model checking can only verify medium-sized systems. On the other hand, theorem proving is not limited in principle by the size, or even by the infinite nature of the system verified, but it is limited in practice by the amount of time, effort and skill that users may invest in a given verification effort.

To overcome these limits, combinations of model checking and theorem proving through *abstraction* and *compositionality* have been advocated [9]. One typical instance of this framework (among others) is the following [10]: first, the PVS theorem prover [8] is employed for building an abstract, finite-state version of the system, which preserves the properties of interest for verification. The theorem prover operates in a compositional manner, i.e. each component of the system is 'abstracted' individually, using little or no information about the other components. Then, the properties are verified at the abstract level using the SMV model checker [11] and, if the verification is successful, the properties hold at the concrete level as well.

In this paper we propose another manner of combining state-space exploration, abstraction, compositional reasoning and theorem proving, and report the application of the method to a real case study.

The case study is the SSCOP (Service Specific Connection Oriented Protocol) [12]. Among other uses, the protocol has been proposed as an alternative to TCP in datagram satellite networks [13].

The SSCOP is a member of the ATM adaptation layer, whose main role is to adapt the unreliable services provided by the lower (physical) layer, to reliable connections and data transfer between two ends. The protocol provides its upper layers with services such as connection establishment and release, error reporting, flow control, using a sliding-window mechanism and secure data transfer, using selective retransmission of protocol data units (PDU). It is standardized in [12], a document consisting of an informal natural-language description and a formal specification in SDL (Specification and Description Language).

The *secure data transfer* service consists of mechanisms for sending PDUs and detecting and retransmitting lost PDUs. It is a complex, infinite-state system that occupies 12 of the 46 pages of the specification [12]. It uses nine integer counters, two queues, two arrays and six lists, all potentially unbounded. The lists contain records with several fields, one of which is again a potentially unbounded list.

*A preliminary version of this paper has appeared in *Formal Methods Europe (FME'03)*, LNCS 2805, pp 223–243. The URL http://www.irisa.fr/vertecs/Equipe/Rusu/sscop contains PVS specifications and proofs for the case study.

The *main property* verified concerns the secure data transfer service between the sender and the receiver's clients. It says that when the transmission ends, the sequence of messages delivered to the receiver's client equals the sequence that the sender's client requested to send. This property is achieved by selective retransmission of lost messages.

Another property that the protocol consistently enforces for efficiency reasons is that no data message is retransmitted unless it was really lost. However, there are two distinct (and distant) mechanisms for requesting retransmissions: one is controlled by the sender and the other one is controlled by the receiver. These two mechanisms have to co-operate in order to avoid useless retransmissions. This intricate behaviour makes the protocol a challenging case study for verification.

1.1. METHODOLOGY

1.1.1. Guided simulation

By examining the SDL specification it can be noted that the protocol does not satisfy its *main property* in all environments. This is confirmed by simulating the specification using the ObjectGeode tool [14]. For example, at any time during a data transmission, the upper layer may emit an 'abort transmission' signal, and then retrieve the messages received so far. Typically, this is only a subsequence of what has been sent, which violates the protocol's *main property*.

Hence, it is *necessary* to assume that some erroneous behaviours of the layers surrounding the protocol, which make the protocol violate its main property, do not occur. The rest of the methodology is employed to prove that excluding such behaviours is also *sufficient* for the protocol to satisfy its main property.

1.1.2. Partial-order abstraction

The second ingredient in our methodology is an abstraction that consists in considering some sequences of actions as *atomic*. This is a well-known technique used in finite-state model checking [15, 16, 17]. It is here adapted to the deductive verification of a class of safety properties on a class of infinite-state systems. Unlike other abstraction techniques such as *predicate abstraction*, e.g. [2, 10, 18] and many others, partial order-abstraction is an *under-approximation* and therefore it is not correct 'by construction' for safety properties. We prove that, under reasonable sufficient conditions, the abstraction is correct. This means that the properties verified at the abstract level will also hold at the concrete level. The sufficient conditions are well adapted for deductive verification (but not for model checking, as checking them may be as hard as checking the main properties of interest) and, to our best knowledge, they are original.

1.1.3. Compositional, deductive verification

For verifying invariants we use PVS [8], a proof assistant based on typed higher-order logic. An automated PVS strategy is used, which attempts to prove that a predicate is inductive, i.e. true initially and preserved by every transition. If this is the case, the predicate is an invariant, otherwise, the subgoals left unproved by PVS suggest auxiliary invariants that, if proved, allow the initial goal to be proved. This technique, called *invariant strengthening*, has been proposed in [19] in the context of deductive protocol verification using PVS.

However, unlike [19] we perform invariant strengthening in a *compositional* manner: each entity (sender and receiver) requires its own set of auxiliary invariants, many of which, thanks again to the *partial-order abstraction*, are preserved by construction by the other entity and do not require a proof. This saves many proof obligations which, without abstraction, would have made the verification infeasible. The compositional rule is proved correct and, to our best knowledge, is also original.

Overall, we obtain a verification methodology for communicating systems, which integrates and adapts several existing verification techniques in a new and meaningful manner. The methodology is effective enough to sucessfully deal with medium-sized system like the SSCOP protocol. The case study took 3 months to complete. We believe this is a reasonable amount of time for verifying a real protocol.

The rest of the paper is organized as follows. In Section 2 a model of extended automata is defined, together with a method (similar to invariant strenghtening [19]) for the verification of invariance properties using PVS. A simplification operation (called *collapsing*) is also defined, which transforms an extended automaton into a smaller, equivalent automaton that is typically easier to verify.

Section 3 deals with the parallel composition of extended automata. Two parallel composition operations are defined: the *interleaved* composition is a usual asynchronous parallel composition; the *atomic* composition lets some particular sequences of transitions be executed atomically. It is shown that, under reasonable sufficient conditions, the verification of an interleaved parallel composition can be reduced to the (typically easier) verification of an atomic parallel composition.

Then, Section 4 provides a compositional rule for the atomic parallel composition. In conjunction with the simplification/reduction techniques defined in the previous sections, the rule typically generates fewer proof obligations than the global invariant strenghtening process introduced in [19].

The resulting methodology has been applied to the SSCOP protocol; the results on this case study are presented in Section 5. Section 6 concludes and discusses related work. For better readability, all the proofs of the results in the paper have been placed in the Appendix.

2. MODELS

In this section the model of extended automata is presented (Section 2.1), together with a method implemented in PVS

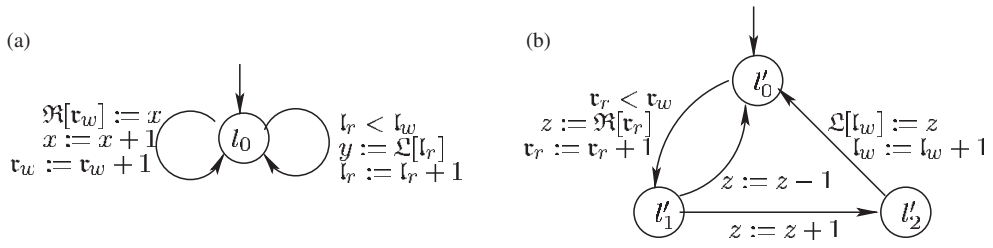


FIGURE 1. (a) sender and (b) receiver.

for proving invariance properties of this model (Section 2.2). The *collapsing* of an extended automaton is defined in Section 2.3.

2.1. Extended automata

DEFINITION 1. (Syntax). *An extended automaton consists of*

- *a finite set of typed variables V*,
- *an initial condition* Θ, *which is a predicate on the variables*,
- *a finite set of control locations L*, *partitioned into two sets of stable locations* L_s, *and a set of unstable locations* $L_u = L \backslash L_s$,
- *a subset* L_i *of initial locations*, *which are all stable*, *i.e.* $L_i \subseteq L_s$,
- *a finite set of transitions* $\mathcal{T}$. *Each transition* $\langle l, G, A, l' \rangle$ *consists of*
  - *— an origin location* $l \in L$,
  - *— a guard G*, *which is a predicate on the variables V*,
  - *— a vector of assignments* $A : (x := e_x)_{x \in V}$, *which contains exactly one assignment* $x := e_x$ *for each variable* $x \in V$. *Here*, e_x *is an expression on the variables V*, *and has same type as x*,
  - *— a destination location* l'

The distinction between stable and unstable locations is important for specification and verification purposes. Intuitively, a system may wait in stable locations for input from the environment; in contrast, unstable locations denote transitory states in which the system only performs internal actions and outputs. This distinction is also present in verification formalisms such as IF [20]. Also, most of the properties verified concern the values of the variables at *stable* locations.

The types of variables include scalar types (integers, enumerated types), and array types, which are functions from the integers to some scalar or other array type.

2.1.1. Notations and conventions

For a variable f of an array type, the expression $f[e_1] := e_2$ is an abbreviation for $f := \lambda k.$ if $k = e_1$ then e_2 else $f(k)$. Intuitively, this means that the new value of f is identical to the old one, except at position e_1, where the value of e_2 is assigned. In the graphical representation of extended automata, variables that are not explicitly present in an assignment are assumed to be unmodified.

EXAMPLE 1. The two automata depicted in Figure 1 describe a simple 'protocol' in which the sender $\mathcal{S}$ and receiver $\mathcal{R}$ exchanges values–via the $\mathfrak{R}$ and $\mathcal{L}$ channels, modeled as shared array variables.

The automaton $\mathcal{S}$ represented in Figure 1 (a) has one location l_0 (which is initial and stable), five integer variables x, y, $\mathfrak{l}_r$, $\mathfrak{l}_w$, $\mathfrak{r}_w$, and two array variables $\mathfrak{R}$ and $\mathcal{L}$. The latter are infinite arrays of integers, i.e. functions from integers to integers. The initial condition (not shown in Figure 1) sets all integer variables to 0. The array variables model potentially unbounded FIFO queues:

- $\mathfrak{R}$ models an output channel for the process $\mathcal{S}$, which writes to $\mathfrak{R}$ the value of x at position $\mathfrak{r}_w$.
- $\mathcal{L}$ models an input channel for the process $\mathcal{S}$, which reads from $\mathcal{L}$ at position $\mathfrak{l}_r$, and places the result in variable y. This may happen only if $\mathfrak{l}_r < \mathfrak{l}_w$, i.e. if there is at least one element in the channel.

Another example of extended automaton is $\mathcal{R}$, depicted in Figure 1 (b). This automaton has three locations, l'_0, l'_1, l'_2. The initial location is l'_0; it is the only stable location. The automaton has four integer variables z, $\mathfrak{r}_r$, $\mathfrak{r}_w$, $\mathfrak{l}_w$ (all set to 0 by the initial condition, not shown in the figure), and two array variables $\mathfrak{R}$ and $\mathcal{L}$ modeling FIFO queues that play roles symmetrical to those in the automaton $\mathcal{S}$, i.e. $\mathfrak{R}$ models an input channel for $\mathcal{R}$, and $\mathcal{L}$ models an output channel.

By composing the two automata according to the usual shared-variable parallel composition we obtain a simple protocol in which the sender $\mathcal{S}$ places the current value of its variable x on the $\mathfrak{R}$ channel. The receiver $\mathcal{R}$ reads this sequence at the other end, and memorizes its reading in variable z, which may be incremented (on the transition from l'_1 to l'_2) or decremented (on the transition from l'_1 to l'_0). Then, $\mathcal{R}$ writes the value of z on the $\mathcal{L}$ channel. The channel is read at the other end by $\mathcal{S}$, which places the result in the variable y.

DEFINITION 2. (Contiguous sequence). *A sequence $\sigma : \tau_1\tau_2\cdots\tau_n$ of transitions of an extended automaton $\mathcal{E}$ is contiguous if for all $i = 1,\ldots,n-1$, the destination of τ_i equals the origin of τ_{i+1}. A contiguous sequence σ is denoted by $\sigma : l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$, where for all $i = 1,\ldots n$, the transition τ_i has the origin l_i as origin and destination l_{i+1}.*

DEFINITION 3. (Syntactical macro-step). *A contiguous sequence $\sigma : l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ of transitions of an extended automaton $\mathcal{E}$ is a syntactical macro-step if the locations l_1 and l_{n+1} are stable, and, for all $j = 2,\ldots,n$, the location l_j is unstable.*

For example, the automaton $\mathcal{R}$ depicted in Figure 1 has one stable location l'_0. The sequences of transitions $l'_0 - l'_1 - l'_2 - l'_0$ and $l'_0 - l'_1 - l'_0$ are syntactical macro-steps.

2.1.2. Semantics

Consider an arbitrary extended automaton with variables V, initial condition Θ, locations $L = L_u \cup L_s$, initial locations $L_i \subseteq L_s$, and transitions $\mathcal{T}$. A *valuation* is a function that associates to each variable $x \in V$ of type T a value $v(x)$ of the same type. For an expression e of type T and a valuation v, $e(v)$ denotes the value of type T obtained by replacing each variable $x \in V$ in e by the value $v(x)$. If the type T is Boolean and $e(v)$ evaluates to *true* then *v satisfies the predicate e*.

For a transition τ with assignments A and v a valuation of the variables, we let $A(v)$ denote the valuation obtained by performing the assignments A on v; i.e. if $A : (x := e_x)_{x\in V}$, then, for each $x \in V$, $A(v)(x) = e_x(v)$.

DEFINITION 4. (States, runs and invariants).

- *A state is a pair $\langle l, v\rangle$ where l is a location and v a valuation. The set of states is denoted by S. A state $\langle l, v\rangle \in S$ is stable if l is a stable location. An initial state is a state whose location l is initial and whose valuation v satisfies the initial condition Θ.*
- *The transition relation ρ_τ of a transition τ with guard G and assignments A is the set of pairs of states $(s, s') \in S \times S$ such that, if $s = \langle l, v\rangle$ and $s' = \langle l', v\rangle'$, then the origin of τ is l, the destination of τ is l', the valuation v satisfies G, and the valuation v' equals $A(v)$. The global transition relation of the automaton is $\rho = \cup_{\tau\in\mathcal{T}}\rho_\tau$.*
- *A run fragment of the automaton is a sequence of states $r : s_1, s_2, \ldots, s_n$ $(n \in \mathbb{N})$ such that there exists a contiguous sequence of transitions $\tau_1, \tau_2, \ldots, \tau_{n-1}$ with $(s_i, s_{i+1}) \in \rho_{\tau i,}$ for $i = 1,\ldots,n-1$. If the sequence σ is is a syntactical macro-step (cf. Definition 3), we say that r is a semantical macro-step. A run is a run fragment that starts in an initial state. A state is reachable if it is the last state of some run.*
- *A predicate P on the states of the automaton $\mathcal{E}$ (also called a state predicate of $\mathcal{E}$) is an invariant, denoted $\mathcal{E} \models \Box P$, if P holds in every reachable state of $\mathcal{E}$.*

2.2. Proving invariants of extended automata using PVS

A transition τ of an extended automaton $\mathcal{E}$ is said to *preserve* a predicate P if, by using only the hypothesis that P is true before the transition τ, it can be inferred that P is true after τ is fired. Formally, let the *weakest precondition* of P by τ be

$$\widetilde{pre}_\tau(P) : \forall s'.\, \rho_\tau(s, s') \Rightarrow P(s') \tag{1}$$

Then, the transition τ preserves P if the implication $P \Rightarrow \widetilde{pre}_\tau(P)$ is valid.

The predicate P is *inductive* if P holds in the initial states of $\mathcal{E}$ and it is preserved by each transition of $\mathcal{E}$. If P is inductive, then P is an invariant, but the converse is not true. This is because the basic hypothesis that P is true before a transition is usually not enough to prove its preservation. Typically, the hypothesis P has to be strengthened using *auxiliary predicates*; we say that *the transition τ preserves P using the auxiliary predicate Q* if, by assuming that $P \wedge Q$ is true before a transition τ, it can be inferred that P is still true after τ (formally, the implication $(P \wedge Q) \Rightarrow \widetilde{pre}_\tau(P)$ is valid).

It is not hard to show that, if Q is an invariant and all transitions of the system preserve P using Q (and P holds in the initial states of the system), then P is an invariant as well. In general, there is no automatic means for computing auxiliary *invariant* predicates Q, because in general the verification of invariants is undecidable.

Our heuristic method implemented in PVS assists the user in the process of discovering such auxiliary predicates. An *ad hoc* PVS proof strategy is employed, which is independent from the system and property under verification. If the strategy is successful, this means that the predicate P under proof is inductive, and in particular, that it is an invariant. But if the strategy fails, PVS presents the user with pending (unproved) subgoals, which suggest auxiliary predicates Q required for proving P[1]. Now, to prove that Q is an invariant may require another predicate R, and the process continues until all the predicates generated in this manner are proved invariant.

This invariant-strenghtening process was proposed in [19] in the context of verifying a data-link protocol using PVS. What is new in our approach is the combination of invariant-strenghtening with partial-order abstraction and with compositionality, which will be presented in the following sections. We first present a simple, yet useful operation that simplifies extended automata prior to verification.

[1] A natural candidate for the auxiliary predicate Q is the predicate $\wedge_{\tau\in T}(P \Rightarrow \widetilde{pre}_\tau(P))$. Our PVS strategy generates a PVS representation of this formula as a number of pending subgoals; typically, one subgoal for each transition $\tau \in T$. The auxiliary predicate Q is obtained by taking the constraints on the pre-states from the generated subgoals.

2.3. Simplifying extended automata: collapsing transitions

The *collapsing* operation amounts to encoding the effect of a contiguous sequence of transitions into one transition. It produces an automaton that typically has fewer unstable locations than the original one. The result is an equivalent automaton, which is typically easier to verify. Equivalence here means that *stable* predicates, defined below, are invariants of the collapsed automaton if and only if they are invariants of the original automaton.

Remember that each transition τ has a guard, which is, syntactically speaking, a predicate on the variables, and a vector of assignments, which, syntactically, is a tuple of assignments (one assignment for each variable). At the semantic level, a guard can be seen as a function from *valuations* (of the variables) to the *Booleans*, and assignments can be seen as functions from *valuations* to *valuations*. By abuse of notation we shall sometimes identify the guard (resp. the assignments) of a transition with their respective semantic functional interpretations.

Hence, assignments can be composed as functions: for assignments A_1, A_2, the assignment $A_2 \circ A_1$ denotes the function that, to each valuation v of the variables, associates the valuation $A_2(A_1(v))$. Also, a guard G and an assignment A can be composed together, i.e. $G \circ A$ denotes the function that associates, to each valuation v, the Boolean value $G(A(v))$.

DEFINITION 5. (Collapsing a finite sequence). *Let* $\sigma : l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ *be a finite, contiguous sequence of transitions of an extended automaton* $\mathcal{E}$, *and, for* $i = 1, \ldots, n$, *let the guard and assignments of* τ_i *be* G_i, A_i. *We define the transition* $\tau = Collapse(\sigma)$ *as follows*: *the origin of* τ *is* l_1; *its destination is* l_{n+1}; *its guard is* $G_1 \wedge (G_2 \circ A_1) \wedge \ldots (G_n \circ A_{n-1} \circ \cdots \circ A_1)$, *and its assignments are* $A_n \circ A_{n-1} \circ \cdots \circ A_1$.

The composition operation $\circ$ can be implemented using syntactic substitution: if A_1 is the assignment $(x := e_x)_{x \in V}$ and A_2 is the assignment $(x := f_x)_{x \in V}$, then, $A_2 \circ A_1$ is the assignment $(x := f_x(x/e_x))_{x \in V}$, where every occurrence of x in the right-hand side expression f_x has been replaced by e_x. Similarly, if G is a guard, then the guard $G \circ A_1$ is obtained by substituting, for every $x \in V$, all the occurrences of x in G by the corresponding expression e_x.

Proposition 1 below says that the effect of the transition $Collapse(\sigma)$ is equivalent to that of σ. There, $\bullet$ denotes the composition of binary relations, defined as follows:

DEFINITION 6. *For a set* X *and two binary relations* α, $\beta \subseteq X \times X$, *the composition* $\beta \bullet \alpha$ *denotes the set* $\{(x, y) \in X \times X | \exists z \in X.(x, z) \in \alpha \wedge (z, y) \in \beta\}$.

PROPOSITION 1. *Let* $\sigma : l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ *be a finite, contiguous sequence of transitions of an extended automaton* $\mathcal{E}$. *Let* ρ_i *denote the transition relation of the transition* τ_i, *for* $i = 1, \ldots, n$, *and* ρ *denote the transition relation of the transition* $\tau = Collapse(\sigma)$. *Then* $\rho = \rho_n \bullet \rho_{n-1} \bullet \cdots \bullet \rho_1$.

Proof. Cf. the Appendix.

We apply the collapsing operation to a finite subset of the automaton's syntactical macro-steps (cf. Definition 3). In general, an extended automaton may contain infinitely many syntactical macro-steps: when a syntactical macro-step contains an unstable location that belongs to a *cycle* of unstable locations in the graph of the automaton, then, by iterating the cycle, an infinite number of syntactical macro-steps can be generated.

DEFINITION 7. (Maximal syntactical macro-step). *Asyntactical macro-step* σ *is maximal if it does not contain a cycle of unstable locations, i.e. a contiguous subsequence* $\sigma' : l_1 \xrightarrow{\tau_1} \ldots l_m \xrightarrow{\tau_m} l_1$ *such that for all* $i = 1, \ldots, m$, l_1 *is unstable*.

For example, in the automaton $\mathcal{R}$ depicted in Figure 1, the sequences of transitions $l'_0 - l'_1 - l'_2 - l'_0$ or $l'_0 - l'_1 - l'_0$ are maximal syntactical macro-steps. On the other hand, e.g. by adding a self-looping transition τ on location l'_2, the macro-step $l'_0 - l'_1 - l'_2 - l'_0$ becomes non-maximal. This is because it is now a subsequence of (infinitely many) syntactical macro-steps, which are all obtained by iterating arbitrarily many times the self-looping transition τ in l'_2.

PROPOSITION 2. *An extended automaton has finitely many maximal syntactical macro-steps.*

The collapsing of an extended automaton $\mathcal{E}$ consists in collapsing the maximal syntactical macro-steps of $\mathcal{E}$, leaving the rest of the automaton unchanged.

DEFINITION 8. (Collapsing an automaton). *For* $\mathcal{E}$, *an extended automaton, the extended automaton* $Collapse(\mathcal{E})$ *is the smallest structure built as follows*:

(i) *the variables and initial condition of* $Collapse(\mathcal{E})$ *are those of* $\mathcal{E}$;
(ii) *for each maximal syntactical macro step* $\sigma : l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ *of* $\mathcal{E}$, $Collapse(\mathcal{E})$ *contains the transition* $Collapse(\sigma)$ (*cf. Definition* 5) *and the locations* l_1 *and* l_{n+1};
(iii) for every non-maximal syntactical macro step $\sigma' : l'_1 \xrightarrow{\tau'_1} l'_2 \cdots l'_m \xrightarrow{\tau'_m} l_{m+1}$, $Collapse(\mathcal{E})$ *contains all the transitions* $\tau'_1, \ldots, \tau'_m$ as well as all the locations $l'_1, \ldots, l'_{m+1}$.

As maximal syntactical macro-steps are finitely many (cf. Proposition 2), the collapsing of an extended automaton can be performed algorithmically. Indeed, Step (ii) of Definition 8 involves a finite set of operations and Step (iii) reaches a fixpoint (i.e. it does not add any new transitions and locations to the resulting automaton) after finitely many operations. For example, for the automaton $\mathcal{R}$ depicted in Figure 1, the automaton $Collapse(\mathcal{R})$ is depicted in Figure 2.

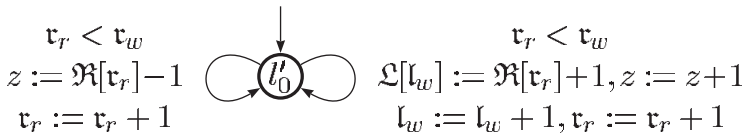


FIGURE 2. Automaton $\mathcal{R}' = Collapse(\mathcal{R})$.

The main property of collapsing is that it preserves reachability of stable states:

PROPOSITION 3. *Let $\mathcal{E}$ be an extended automaton. Then, a stable state is reachable in $\mathcal{E}$ if and only if it is also reachable in Collapse($\mathcal{E}$).*

2.3.1. Stable predicates

Most of the verification techniques defined in this paper are tailored to the class of *stable* predicates, defined below. Intuitively, the truth value of such a predicate is only relevant in the stable states of a system, but it is trivially *true* in the unstable states. In the sequel, we assume that extended automata have a dedicated variable *pc* ranging over locations; hence, for l a location of the automaton, $pc = l$ is a predicate denoting the fact that control is currently at location l.

DEFINITION 9. *A state predicate P of an extended automaton $\mathcal{E}$ is stable if P is of the form $(pc = l) \Rightarrow P'$, where l is a stable location and P' is a state predicate.*

Then, proving the invariance of a stable predicate on an extended automaton is equivalent to (but, typically, easier than) proving it on the corresponding collapsed automaton:

PROPOSITION 4. *If P is a stable predicate of $\mathcal{E}$, then $Collapse(\mathcal{E}) \models \Box P$ iff $\mathcal{E} \models \Box P$.*

It turns out that the invariant-strengthening process described in Section 2.2 typically generates auxiliary proof obligations of the form $Q : (pc = l) \Rightarrow Q'$, where $pc = l$ is a predicate characterising presence at location l and Q' is some state predicate. Collapsing reduces the number of locations and, consequently, the number of proof obligations. Hence, proving $Collapse(\mathcal{R}) \models \Box\varphi$ is easier than proving $\mathcal{R} \models \Box\varphi$, and the benefits increase with the number of sequences that are collapsed.

3. COMPOSING EXTENDED AUTOMATA

The techniques presented so far allow us to verify systems modeled using one global extended automaton. However, many systems consists of several communicating entities, which are best modeled using parallel compositions of extended automata.

Section 3.1 defines two parallel composition operations. The first one, *interleaved* parallel composition, is more realistic from a specification point of view; but the second one, *atomic* parallel composition, is more practical for verification. We prove in Section 3.2 that, under reasonable sufficient conditions, a property verified under the atomic parallel composition holds under the interleaved parallel composition.

It should be noted that in other application domains, such as hardware, neither of the above essentially *asynchronous* compositions is realistic. A different, *synchronous* composition is typically used there.

3.1. Interleaved and atomic parallel compositions

The *interleaved* parallel composition, denoted by $\|$, is the usual asynchronous parallel composition of programs with shared variables. The *atomic* parallel composition, denoted by $|$, allows some particular sequences of steps to be executed atomically.

3.1.1. Interleaved parallel composition

The initial location of the *interleaved* parallel composition $\mathcal{E}_1\|\mathcal{E}_2$ is the set of pairs consisting of initial locations of $\mathcal{E}_1$ and of $\mathcal{E}_2$, and the initial condition is the conjunction of those of $\mathcal{E}_1$, $\mathcal{E}_2$. A state s of the composed system is a pair (s_1, s_2) of states of the components that agree on the values of the common variables. Formally, let $v^{\downarrow V'}$ denote the restriction of a valuation v to a subset $V' \subseteq V$ of variables. The states $s_1 = \langle l_1, v_1\rangle$ of $\mathcal{E}_1$ and $s_2 = \langle l_2, v_2\rangle$ of $\mathcal{E}_2$ constitute a state $s = (s_1, s_2)$ of $\mathcal{E}_1\|\mathcal{E}_2$ if $v_1^{\downarrow V_1\cap V_2} = v_2^{\downarrow V_1\cap V_2}$, where V_1, V_2 denote the sets of variables of $\mathcal{E}_1$,$\mathcal{E}_2$. For a state $s = \langle l, v\rangle$, we denote by $s^{\downarrow V'}$ the pair $\langle l, v^{\downarrow V'}\rangle$. The transition relation ρ of $\mathcal{E}_1\|\mathcal{E}_2$ holds for states $s = (s_1, s_2)$ and $s = (s'_1, s'_2)$ of $\mathcal{E}_1\|\mathcal{E}_2$ if either $s_1'^{\downarrow V_1\setminus V_2} = s_1^{\downarrow V_1\setminus V_2}$ and $\rho_2(s_2, s'_2)$ holds, or $s_2^{\downarrow V_2\setminus V_1} = s_2'^{\downarrow V_2\setminus V_1}$ and $\rho_1(s_1, s'_1)$ holds, where ρ_1 (resp. ρ_2) is the transition relation of $\mathcal{E}_1$ (resp. $\mathcal{E}_2$).

3.1.2. Atomic parallel composition

The initial location, initial condition, and states of the atomic parallel composition $\mathcal{E}_1|\mathcal{E}_2$ are defined just like for $\mathcal{E}_1\|\mathcal{E}_2$. Remember that a (semantical) *macro-step* (cf. Definition 4) is a run fragment $s^1, \ldots, s^k$, where $s^i = \langle l^i, v^i\rangle$ for $i = 1, \ldots, k$ such that l^1 and l^k are stable and for $j = 2, \ldots k-1$, l^j is unstable. Let the *atomic* transition relation ϱ_1 of A_1 (and similarly, ϱ_2 for A_2) be the set of pairs of states (s, s') for which there exists a macro-step between s and s'. Then, the

transition relation ϱ of the composed system $\mathcal{E}_1|\mathcal{E}_2$ holds for states $s = (s_1, s_2)$ and $s' = (s'_1, s'_2)$ if either $s_1'^{\downarrow V_1 \setminus V_2} = s_1^{\downarrow V_1 \setminus V_2}$ and $\varrho_2(s_2, s'_2)$ holds, or $s_2'^{\downarrow V_1 \setminus V_2} = s_2^{\downarrow V_2 \setminus V_1}$ and $\varrho_1(s_1, s'_1)$ holds.

3.1.3. Interpretation

The atomic parallel composition allows one component to move by performing a whole macro-step, while the other component does not move. For example, remember that l'_0 is the only stable location of the automaton $\mathcal{R}$ depicted in Figure 1. Then, e.g. $\mathcal{R}$ may perform the sequences of transitions $l'_0 - l'_1 - l'_2 - l'_0$ or $l'_0 - l'_1 - l'_0$ without the automaton $\mathcal{S}$ performing any transition in between.

While convenient for verification purposes, the atomic parallel composition is not realistic for modeling distant communicating entities, because it cannot be reasonably assumed that one entity, at one end of the protocol, performs a potentially unbounded sequence of actions, while the entity at the other end does not move.

3.2. Abstraction: from interleaved to atomic composition

In this section we show that, under reasonable sufficient conditions, the verification of automata composed using the interleaved ($\|$) parallel composition reduces to that of |-composed systems, which can be dealt with compositionally (cf. Section 4).

First, a notion of *independence* between transitions of extended automata is defined, and sufficient conditions are given, which can be used in practice for establishing the independence of two transitions. The notion of independence is then generalized to extended automata, together with sufficient conditions for independence, and a heuristic for helping to establish the independence of extended automata that model communication protocols. We then show that, for *mutually independent* extended automata $\mathcal{E}_1$, $\mathcal{E}_2$ and the class of *stable* predicates P (cf. Definition 9), $\mathcal{E}_1\|\mathcal{E}_2 \models \Box P$ holds if and only if $\mathcal{E}_1|\mathcal{E}_2 \models \Box P$ holds. For τ a transition and s, s' states of an extended automaton, we write $s \xrightarrow{\tau} s'$ for $(s, s') \in \rho_\tau$. Also, if $s = (s_1, s_2)$ and $s' = (s'_1, s'_2)$ are state of the parallel composition $\mathcal{E}_1\|\mathcal{E}_2$ and τ is a transition of $\mathcal{E}_1$, we write $s \xrightarrow{\tau} s'$ as an abbreviation for $(s_1, s'_1) \in \rho_\tau$ and $s_2 = s'_2$.

DEFINITION 10. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be extended automata, τ_1 a transition of $\mathcal{E}_1$, τ_2 a transition of $\mathcal{E}_2$, and s a state of $\mathcal{E}_1\|\mathcal{E}_2$. We say that τ_2 is independent of τ_1 in state s if for all states $s', s'' : s \xrightarrow{\tau_1} s' \xrightarrow{\tau_2} s''$ implies that there exists a state $\tilde{s}$ such that $s \xrightarrow{\tau_2} \tilde{s} \xrightarrow{\tau_1} s''$. If τ_2 is independent from τ_1 and τ_1 is independent from τ_2 in state s, we say that τ_1 and τ_2 are mutually independent in state s.*

If τ_2 is independent of τ_1 in s, the execution of the sequence $\tau_1\tau_2$ in s can be replaced by $\tau_2\tau_1$, which produces the same global effect. Similar properties [15, 16, 17] are used in model checkers [4, 20] to fight the state-space explosion problem. In our theorem-proving framework, independence of transitions is employed for reducing the number of auxiliary proof obligations required for proving a given property.

Proposition 5 below gives sufficient conditions for independence in reachable states of $\mathcal{E}_1\|\mathcal{E}_2$. We use the following notations: let τ be a transition of an extended automaton $\mathcal{E}$, and A denote its vector of assignments $(x := e_x)_{x \in V}$; we denote by $rhs(A)$ the vector of expressions $(e_x)_{x \in V}$. For τ_1, τ_2 two transitions of an automaton $\mathcal{E}$ and $A_1 : (x := e_x)_{x \in V}$, $A_2 : (x := f_x)_{x \in V}$ their respective vectors of assignments, remember (cf. Section 2.3) that $A_1 \circ A_2$ denotes the assignments $(x := e_x(x/f_x))$, whose right-hand sides e_x have every occurrence of x replaced by f_x. The equality of vectors of expressions, e.g. $(e_x)_{x \in V} = (f_x)_{x \in V}$ is an abbreviation for the set of equalities $e_x = f_x$, for all $x \in V$.

Finally, the dual *pre* of the weakest pre-condition $\widetilde{pre}$ [cf. Equation (1)] is

$$\mathrm{pre}_\tau(P) : \exists s' \cdot \rho_\tau(s, s') \wedge P(s') \tag{2}$$

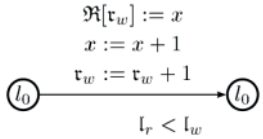
PROPOSITION 5. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be extended automata, τ_1 a transition of $\mathcal{E}_1$, and τ_2 a transition of $\mathcal{E}_2$. Let G_1 denote the guard of τ_1, l_1 denote the origin of τ_1, and A_1, A_2 denote respectively the assignments of τ_1, τ_2. Then, τ_2 is independent of τ_1 in every reachable state of $\mathcal{E}_1\|\mathcal{E}_2$ if*

(i) $\mathcal{E}_1\|\mathcal{E}_2 \models \Box[(pc_1 = l_1 \wedge G_1) \Rightarrow pre_{\tau_2}(G_1)]$ *and*
(ii) $\mathcal{E}_1\|\mathcal{E}_2 \models \Box[(pc_1 = l_1 \wedge G_1) \Rightarrow rhs(A_1 \circ A_2) = rhs(A_2 \circ A_1)]$

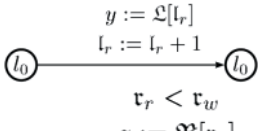
Conditions (i) and (ii) in the Proposition are undecidable for arbitrary extended automata, but we are here in a deductive verification framework, where the main properties of interest are also undecidable. Hence, Proposition 5 is useful in practice when the verification of conditions (i) and (ii) does not generate too much verification overhead. This works well when the invariants (1) and (2) hold 'by construction', (e.g. when the transitions τ_1, τ_2 access disjoint sets of variables, or, in the case of communication protocols, when τ_2 writes at one end of a channel while τ_1 reads from the other end of the same channel).

EXAMPLE 2. Consider the extended automata depicted in Figure 1. We label the transitions as follows:

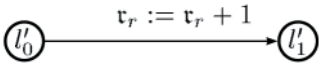
1. τ_1 denotes:

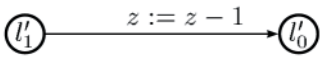


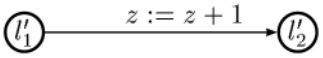
and τ_2 denotes:

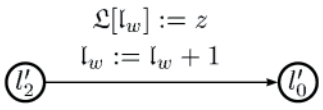


2. τ_3 denotes:



and τ_4 denotes: 

3. τ_5 denotes: 

and τ_6 denotes: 

We show using Property 2 that τ_1 is independent of τ_3 in all reachable states of $\mathcal{S}\|\mathcal{R}$. The underlying reason is that τ_1 writes at one end of the channel $\mathfrak{R}$, while τ_3 reads from the same channel at the other end.

Hypothesis (2) is easy to check, as all the variables that are syntactically modified by τ_1 (i.e. $\mathfrak{R}$, x and $\mathfrak{r}_w$) are syntactically not modified by τ_3, and reciprocally.

We now check Hypothesis (1). The guard of τ_3 is $G_3 : \mathfrak{r}_r < \mathfrak{r}_w$. Then, we have $\text{pre}_{\tau 1}(G_3)$: $\mathfrak{r}_r \leq \mathfrak{r}_w$.

It is not hard to check that the predicate $\mathfrak{r}_r \leq \mathfrak{r}_w$ is preserved by all the transitions of $\mathcal{S}$ (τ_1, τ_2) and of $\mathcal{R}$ $(\tau_3, \ldots, \tau_6)$. Hence, this predicate $[\text{pre}_{\tau_1}(G_3)]$ is also an invariant of $\mathcal{S}\|\mathcal{R}$, which implies Hypothesis (1). Then, by Proposition 5, τ_1 is independent of τ_3 in all reachable states of $\mathcal{S}\|\mathcal{R}$. It can be shown in a similar manner that τ_6 is independent of τ_2; the underlying reason is again that τ_6 writes at one end of a a channel (here, $\mathcal{L}$) while τ_2 reads from the same channel at the other end.

On the other hand, τ_1 is mutually independent with τ_4, τ_5, τ_6; and τ_2 is mutually independent with τ_3, τ_4, τ_5. These mutual independences are proved directly using a syntactical argument: the transitions operate on disjoint sets of variables.

Finally, τ_3 is not independent of τ_1 in the initial states (where $\mathfrak{r}_r = \mathfrak{r}_w = 0$): the sequence $\tau_1\tau_3$ can be executed, but the sequence $\tau_3\tau_1$ cannot, as the guard $G_3 : \mathfrak{r}_r < \mathfrak{r}_w$ is false.

DEFINITION 11. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be extended automata. We say $\mathcal{E}_1$ is independent of $\mathcal{E}_2$ if every syntactical macro-step σ of $\mathcal{E}_1$ can be decomposed as $\sigma = \sigma' \cdot \sigma''$, such that*

(i) *σ' is either the empty sequence, or it consists of exactly one transition and*
(ii) *σ'' is a (possibly empty) sequence of transitions $\tau''_1 \cdots t''_k$ $(k \in \mathbb{N})$ such that for each $i = 1, \ldots, k$ the transition τ''_i is independent of every transition τ of $\mathcal{E}_2$, in every reachable state of $\mathcal{E}_1\|\mathcal{E}_2$.*

If $\mathcal{E}_1$ is independent of $\mathcal{E}_2$ and $\mathcal{E}_2$ is independent of $\mathcal{E}_1$ we say $\mathcal{E}_1$ and $\mathcal{E}_2$ are mutually independent.

3.2.1. *Heuristic for proving independence*

For communication protocols modeled by extended automata that communicate through uni-directional FIFO channels (which we model using array variables) such as the example depicted in Figure 1, or the SSCOP protocol described in Section 5, the following decomposition $\sigma = \sigma' \cdot \sigma''$ of syntactical macro-steps (cf. Definition 11) can be tried:

- σ' is empty, or it consists of one transition that performs an *input* from one channel,
- σ'' is an arbitrary sequence of *internal actions* and of *outputs* to the other channel.

Indeed, Definition 11 requires that every transition (except maybe the first one) of every syntactical macro-step of one component, to be independent of all the transitions of the other component; and internal actions and outputs typically satisfy this condition. Such heuristics are employed in, e.g. SDL state-exploration tools [14], without any justification as to whether the resulting reduced state-space is correct.

EXAMPLE 3. We show that the automata $\mathcal{S}$ and $\mathcal{R}$ are mutually independent using the above heuristic. The automata have only one stable location each, which corresponds to their respective initial location. With their transitions labeled $\tau_1 \ldots \tau_6$ as in Example 2, $\mathcal{S}$ has two syntactical macro-steps, denoted by $\sigma^1_1 : l_0 \xrightarrow{\tau_1} l_0$ and $\sigma^1_2 : l_0 \xrightarrow{\tau_2} l_0$; and $\mathcal{R}$ also has two syntactical macro-steps, $\sigma^2_1 : l'_0 \xrightarrow{\tau_3} l'_1 \xrightarrow{\tau_4} l'_0$ and $\sigma^2_2 : l'_0 \xrightarrow{\tau_3} l'_1 \xrightarrow{\tau_5} l'_2 \xrightarrow{\tau_6} l'_0$. Consider the following decompositions of these sequences:

- $\sigma^1_1 = \sigma'^1_1 \cdot \sigma''^1_1$ with $\sigma'^1_1 = \varepsilon$ and $\sigma''^1_1 = l_0 \xrightarrow{\tau_1} l_0$;
- $\sigma^2_1 = \sigma'^2_1 \cdot \sigma''^2_1$ with $\sigma'^2_1 = l_0 \xrightarrow{\tau_2} l_0$ and $\sigma''^2_1 = \varepsilon$;
- $\sigma^1_2 = \sigma'^2_1 \cdot \sigma''^2_1$ with $\sigma'^2_1 = l'_0 \xrightarrow{\tau_3} l'_1$ and $\sigma''^2_1 = l'_1 \xrightarrow{\tau_4} l'_0$
- $\sigma^2_2 = \sigma'^2_2 \cdot \sigma''^2_2$ with $\sigma'^2_2 = l'_0 \xrightarrow{\tau_3} l'_1$ and $\sigma''^2_2 = l'_1 \xrightarrow{\tau_5} l'_2 \xrightarrow{\tau_6} l'_0$.

Using the independence relations between transitions $\tau_1 \ldots \tau_6$ that have been established in Example 2, it can be easily shown that these decompositions satisfy Definition 2, i.e. the automata $\mathcal{S}$ and $\mathcal{R}$ from Figure 1 are mutually independent.

DEFINITION 12. *For two extended automata $\mathcal{E}_1$, $\mathcal{E}_2$, a state (s_1, s_2) of $\mathcal{E}_1\|\mathcal{E}_2$ is stable if s_1 is a stable state of $\mathcal{E}_1$ and s_2 is a stable state of $\mathcal{E}_2$. The projection on $\mathcal{E}_1$ of a state $s = (s_1, s_2)$ is defined to be the state s_1, and similarly for $\mathcal{E}_2$.*

PROPOSITION 6. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be mutually independent extended automata, and s a reachable, stable state of $\mathcal{E}_1\|\mathcal{E}_2$. Then, s is reachable in $\mathcal{E}_1|\mathcal{E}_2$.*

Finally, for the class of *stable* predicates (cf. Definition 9), the verification on the interleaved parallel composition is equivalent to that on the atomic parallel composition, provided that the two automata are mutually independent:

PROPOSTION 7. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be mutually independent extended automata and P a stable predicate of $\mathcal{E}_1\|\mathcal{E}_2$. Then $\mathcal{E}_1|\mathcal{E}_2 \models \Box P$ holds if and only if $\mathcal{E}_1\|\mathcal{E}_2 \models \Box P$ holds.*

4. COMPOSITIONAL VERIFICATION

In Section 3.2 it has been shown that, for stable invariants and mutually independent extended automata, it is enough to perform the verification on the *abstract*, atomic parallel composition for the invariants to hold on the interleaved parallel composition. In this section we provide a compositional rule for verifying invariants at the abstract level, and show that this rule can be combined with the collapsing operation defined in Section 2.3 for reducing the number of proof obligations.

The compositional rule (cf. Proposition 4 below) says essentially that the |-composition of two automata satisfies a given invariant whenever the two automata, suitably modified, satisfy the invariant. The precise definition of modified automaton is given below.

DEFINITION 13. (Modified automaton). *Let $\mathcal{E}$ be an extended automaton, P a state predicate of $\mathcal{E}$, and L a set of locations of $\mathcal{E}$. The modified automaton $\mathcal{E}$ **init**$\langle P, L\rangle$ is an automaton identical to $\mathcal{E}$, except that its initial condition is P and its set of initial locations is L.*

For example, for the automaton $\mathcal{S}$ depicted in Figure 1, $\mathcal{S}$ **init**$\langle x \geq y, \{l_0\}\rangle$ has the same locations, variables and transitions as $\mathcal{S}$; its set of stable locations is $\{l_0\}$, which coincides here with that of $\mathcal{S}$; and its initial condition is $x \geq y$, whereas that of $\mathcal{S}$ is more restrictive.

PROPOSITION 8. (Compositional rule). *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be extended automata over a common V set of variables*[2]*, and P a predicate on the states of $\mathcal{E}_1|\mathcal{E}_2$ such that P holds in the initial states of $\mathcal{E}_1$, $\mathcal{E}_2$. Let L_s^1, L_s^2 denote the sets of stable locations of $\mathcal{E}_1$, $\mathcal{E}_2$, respectively.*

If $\mathcal{E}_1$ **init**$\langle P, L_s^1\rangle \models \Box P$ and $\mathcal{E}_2$ **init**$\langle P, L_s^2\rangle \models \Box P$ hold, then $\mathcal{E}_1|\mathcal{E}_2 \models \Box P$ holds.

4.1. Using the compositional rule

By Proposition 8, for proving that a state predicate P is an invariant of $\mathcal{E}_1|\mathcal{E}_2$, it is enough to prove that P holds initially in $\mathcal{E}_1$, $\mathcal{E}_2$, and that P is an invariant of $\mathcal{E}_1$ **init** $\langle P, L_s^1\rangle$ and of $\mathcal{E}_2$ **init** $\langle P, L_s^2\rangle$. The latter being stand-alone components, the basic invariant-strengthening technique (cf. Section 2.2) can now be used. Again, the basic hypothesis that P is true before each transition is usually too weak, and the hypothesis has to be strengthened using auxiliary predicates suggested by PVS.

The difference to proving a stand-alone system is that proof obligations may 'travel' between components: an auxiliary invariant Q used for proving P on one component may also have to be proved preserved by the other. To see this, assume that we want to prove $\mathcal{E}_1|\mathcal{E}_2 \models \Box P$ using Proposition 8. This requires to prove $\mathcal{E}_1$ **init** $\langle P, L_s^1\rangle \models \Box P$; assume P is not inductive, and that PVS suggests an auxiliary invariant Q. Then, even if one proves that $\mathcal{E}_1$ **init** $\langle P \wedge Q, L_s^1\rangle \models \Box(P \wedge Q)$ holds (and Q holds initially in $\mathcal{E}_1$), in order to use Proposition 8, one still has to prove that Q holds initially in $\mathcal{E}_2$, and that the second component preserves Q, i.e. $\mathcal{E}_2$ **init** $\langle P \wedge Q, L_s^2\rangle \models \Box(P \wedge Q)$ holds. Only then can Proposition 8 be used to infer $\mathcal{E}_1|\mathcal{E}_2 \models \Box(P \wedge Q)$, and, in particular, that $\mathcal{E}_1|\mathcal{E}_2 \models \Box P$ holds.

The benefits of the using the compositional rule (over a direct verification of $\mathcal{E}_1\|\mathcal{E}_2 \models \Box P$ by invariant strengthening, cf. Section 2.2) is that some proof obligations can be saved:

- First, if Q is a proof obligation of the form $Q : (pc_1 = l) \Rightarrow Q'$, where $l \in L_u^1$ is an *unstable* location of $\mathcal{E}_1$, then Q is trivially preserved by $\mathcal{E}_2$, because in the *atomic* composition $\mathcal{E}_1|\mathcal{E}_2$, when one component is in an unstable location, the other one does not move. Formally, Q trivially holds in the initial states of $\mathcal{E}_1$ **init** $\langle P, L_s^1\rangle$. Thus, proving $\mathcal{E}_1$ **init** $\langle P \wedge Q, L_s^1\rangle \models \Box(P \wedge Q)$ reduces to proving $\mathcal{E}_1$ **init** $\langle P, L_s^1\rangle \models \Box(P \wedge Q)$. Hence, if one also proves $\mathcal{E}_2$ **init** $\langle P, L_s^2\rangle \models \Box P$ (and P holds initially in $\mathcal{E}_2$) then the compositional rule (Proposition 4) can be used to deduce $\mathcal{E}_1 \mid \mathcal{E}_2 \models \Box P$; thus, a proof that Q is preserved by $\mathcal{E}_2$ has been saved.
- Second, a proof obligation which only involves variables that are *not modified* by a component are trivially preserved by that component. Formally, if Q is an auxiliary invariant that involves only variables not modified by $\mathcal{E}_2$, then $\mathcal{E}_2$ **init** $\langle P \wedge Q, L_s^2\rangle \models \Box Q$ is trivially true, and proving $\mathcal{E}_2$ **init** $\langle P \wedge Q, L_s^2\rangle \models \Box(P \wedge Q)$ reduces to proving $\mathcal{E}_2$ **init** $\langle P \wedge Q, L_s^2\rangle \models \Box P$. Again, a proof that Q is preserved by $\mathcal{E}_2$ was saved.

For example, the data transfer of the SSCOP protocol is decomposed in five components (three for the sender, and two for the receiver, cf. Section 5). Its verification consisted in proving 233 invariants, out of which 47 needed to be proved only on one component out of five; and, among the remaining ones, many more required a proof only on two or three components. In a global verification, each of these invariants would have required a proof on the whole system, which would have generated many more auxiliary proof obligations and would have made the verification practically unfeasible.

4.2. Combining the compositional rule and the collapsing operation

We show that for stable predicates, the compositional rule (Prop. 8) can be equivalently applied to the *collapsed*

[2]This hypothesis is required for the *modified automata* $\mathcal{E}_1$ **init**$\langle P, L_s^1\rangle$ and $\mathcal{E}_2$ **init**$\langle P, L_s^2\rangle$ to be well defined. It can always be assumed, if $\mathcal{E}_1$ and $\mathcal{E}_2$ had different sets of variables V_1, V_2, just take $V = V_1 \cup V_2$. Of course, some of the variables in V_2 will never be actually read or modified by $\mathcal{E}_1$ (more precisely, they will have identity assignments), but formally this is not a problem.

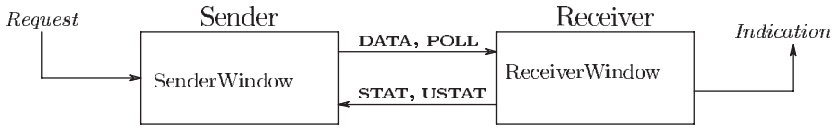


FIGURE 3. The SSCOP Protocol.

automata. The collapsing operation has been defined in Section 2.3. First, the following proposition says that the *Collapse* operation distributes over the | parallel composition:

PROPOSITION 9. (*Collapse* distributes over |). *If P is a stable state predicate of $\mathcal{E}_1|\mathcal{E}_2$, then Collapse $(\mathcal{E}_1|\mathcal{E}_2) \models \Box P$ if and only if Collapse$(\mathcal{E}_1)$|Collapse$(\mathcal{E}_2) \models \Box P$.*

Now, by Proposition 4, $\mathcal{E}_1|\mathcal{E}_1 \models \Box P$ is equivalent to Collapse$(\mathcal{E}_1|\mathcal{E}_2) \models \Box P$ which, by Proposition 9 is equivalent to Collapse$(\mathcal{E}_1)$|Collapse$(\mathcal{E}_2) \models \Box P$. Hence, the compositional rule (Proposition 8) says that, in order to prove $\mathcal{E}_1|\mathcal{E}_1 \models \Box P$ it is enough to show that P holds initially in Collapse$(\mathcal{E}_1)$ and in Collapse$(\mathcal{E}_2)$, and that Collapse$(\mathcal{E}_1)$ **init**$\langle P, L_s^1\rangle \models \Box P$ and Collapse$(\mathcal{E}_2)$**init**$\langle P, L_s^2\rangle \models \Box P$ hold. Hence, $\mathcal{E}_1$ and $\mathcal{E}_2$ can be collapsed prior to using the compositional rule. Our experience with the SSCOP protocol showed that this significantly reduces the number of proof obligations required to complete a proof. As any compositional approach, ours works best for weakly coupled components—like the sender and receiver in communication protocols—and is presumably less effective for more tightly synchronized systems such as hardware.

4.3. Summary: verification methodology

We now combine the results obtained in Sections 2–4 into the following verification methodology. Our goal is to establish $\mathcal{E}_1\|\mathcal{E}_2 \models \Box P$, for P a stable predicate of $\mathcal{E}_1\|\mathcal{E}_2$. First, the automata are checked for mutual independence using Proposition 2 and/or simple syntactical arguments (i.e. the transitions access disjoint sets of variables). Then, the automata are collapsed, and PVS and the invariant-strengthening techniques described in Section 2.2 are used to establish that P holds initially in *Collapse*$(\mathcal{E}_1)$ and in *Collapse*$(\mathcal{E}_2)$, and that *Collapse*$(\mathcal{E}_1)$**init**$\langle P, L_s^1\rangle \models \Box P$ and *Collapse*$(\mathcal{E}_2)$**init**$\langle P, L_s^2\rangle \models \Box P$ hold. By the compositional rule (Proposition 8) this implies that *Collapse*$(\mathcal{E}_1)|$ *Collapse*$(\mathcal{E}_2) \models \Box P$. By Propositions 4 and 9, we then obtain $\mathcal{E}_1\|E_2 \models \Box P$. Finally, Proposition 7 implies that $\mathcal{E}_1\|\mathcal{E}_2 \models \Box P$ holds.

5. VERIFYING THE SSCOP PROTOCOL

In this section the verification of the SSCOP protocol using the approach presented in the previous section is reported. Both the protocol and its verification are quite large (∼1000 PVS lines for the specification, 2000 lines for the 233 theorems, and 23,000 PVS proof commands). They are here described very briefly; see [12, 21, 22] for more details on the system and its verification. We include a small part of the verification to illustrate how the method presented scales up to a real system.

The SSCOP protocol consists of a sender and a receiver that exchange PDUs through communication channels. Let SR designate the channel from the sender to the receiver, and RS the channel from receiver to sender (cf. Figure 3). We here assume that the the initial connection between sender and receiver has been established (the connection process is not modeled here; see, e.g. [21] for the verification of this process by model checking). The sender and receiver are in their secure data transfer modes, which involve four kinds of PDUs.

- DATA PDUs transit on the SR channel. They convey the actual data from sender to receiver. Each DATA PDU also carries a unique index number,
- POLL PDUs also transit on the SR channel. They are used by the sender to question the receiver on its status with regards to reception of DATA PDUs;
- STAT (status) PDUs transit on the RS channel. The receiver replies with a STAT to a POLL of the sender, by reporting a list of DATA PDU that are missing;
- USTAT (unsolicited status) PDUs also transit on the RS channel. They are similar to STAT, except that they are emitted spontaneously by the receiver, when a DATA PDU with a higher index number than expected is received, which means some DATA PDU were lost during the transmission.

The main property verified concerns an arbitrary sequence of messages to be transmitted between the sender and the receiver's clients. The property says that when transmission ends, the *Indication* and *Request* sequences are equal. As the channels are imperfect, the protocol achieves this through selective retransmissions of DATA PDUs, using a mechanism involving POLL, STAT and USTAT PDUs.

Moreover, the protocol does not satisfy this property in all environments. This can be noted by carefully examining the SDL specification [12]; a simulation of the specification using the ObjectGeode tool [14] confirms this observation. Hypotheses on the behaviour of the upper and lower layer have to be made—see below. Note that the simulation only shows that the hypotheses are *necessary*; the rest of the verification amounts to proving that these

hypotheses are also *sufficient* for the protocol to satisfy the main property.

5.1. Hypotheses on surrounding layers

Some behaviours of the upper layer may make the protocol violate its main property. For example, the receiver's client may require disconnection, which causes the transmission to terminate early; when this happens, a state is entered where the incomplete sequence of data received so far may be retrieved by the receiver's client. This violates the protocol's main property.

ASSUMPTION 1. *The upper layer does not disrupt an ongoing data transfer*

Assumptions have to be made on the lower layer as well: the communication channels may only lose, but not create, duplicate, or reorder PDUs. For example, if a DATA PDU reaches the receiver with the same sequence number as one that was already received (duplication), then the protocol exits its data transfer mode and transmission is terminated. Similarly, if two DATA PDU are reordered, the arrival of the first one may trigger the retransmission of the second one; the latter then reaches the receiver, with the same consequence that the transmission is abruptly terminated and the state where the incomplete sequence of messages is retrieved by the receiver's client, again violating the protocol's main property.

ASSUMPTION 2. *The lower layer may lose, but cannot create, duplicate or reorder PDUs*

Finally, the protocol cannot tolerate any amount of losses. A timer expires when no PDUs have arrived for a certain amount of time, which abruptly terminates the transmission and violates the main property:

ASSUMPTION 3. *The connection is not broken because of too many lost PDUs*

5.2. Formally modeling the protocol

The protocol is SSCOP $\coloneqq$ *Sender*$\|$*Receiver*, where $\|$ is the *interleaved* parallel composition (cf. Section 2). This is consistent with the semantics of the protocol originally expressed in SDL [12], in which distant processes may interleave their actions. The lossy channels are modeled using shared array variables, just as in our running example (cf. Sections 2–4).

A brief description of the sender and receiver processes follows. Both processes can be decomposed into several components, which correspond to SDL 'transitions' in the specification [12]. An SDL transition is similar to our notion of macro-step: e.g. performing an input from a channel, followed by arbitrarily many internal actions and outputs to another channel.

The sender process consists of one component for sending DATA and POLL PDUs, another one for receiving USTAT and a third one for receiving STAT PDUs. According to the semantics of SDL, only one of these components can be executing at one time, while the others are waiting in stable locations. This is because all three components belong to the same process, which is assumed to be executing on one single processor. Thus, the sender is defined as the *atomic* composition Sender $\coloneqq$ (DATA & POLL *sender* | USTAT receiver | STAT receiver). The receiver is modeled as the atomic composition of two components, *Receiver* $\coloneqq$ (DATA receiver | POLL receiver).

5.3. Verification: the main invariants

The protocol's main property is expressed using a *stable* predicate (cf. Definition 9), saying that whenever both sender and receiver are in their initial locations, the *Indication* and *Request* sequences are equal. To take advantage of the verification methodology presented in Section 4, the interleaved parallel composition has to be replaced by an atomic one. For this, the independence hypothesis of *Sender* and *Receiver* (cf. Definition 11) has to be satisfied. It is here checked using syntactical arguments and simple invariants, in a manner completely similar to what was described in Section 3.2 (cf. Examples 2 and 3) for the running example of the paper[3]. The system to be verified becomes DATA & POLL *sender* | USTAT *receiver* |STAT *receiver* |DATA *receiver* |POLL *receiver*).

This decomposes each proof in five independent parts, which are dealt with in a compositional manner as shown in Section 4.

5.3.1. *Verifying one representative invariant*

The most important auxiliary property in the whole verification process is that DATA PDUs are not retransmitted unless they were really lost. The mechanism for requesting retransmissions is distributed: the USTAT PDUs are spontaneously generated by the receiver, while STAT PDUs are generated by the receiver in response to a POLL PDU from the sender. These mechanisms have to interact while preserving the given auxiliary property.

To prove the property that DATA PDUs are not retransmitted unless they were really lost, one has to establish, among others:

(i) *Every* DATA *identified as lost by a* STAT *in the* RS *channel, is either lost, or will never be retransmitted.*

In the remainder of the section, we show how to formalize and prove the property (i). The property is about STAT PDUs, which are generated by the POLL receiver component

[3]A total of 28 simple invariants had to be proved, which correspond to using Proposition 5 for showing that transitions writing to one channel are independent of transitions reading from the same channel. All are inductive and are automatically proved by our PVS strategy described in Section 2.2.

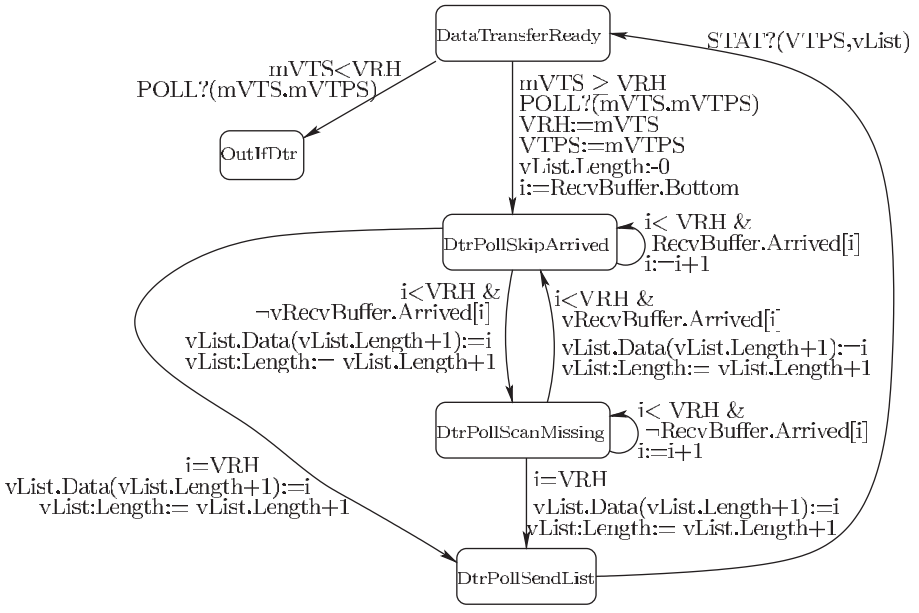


FIGURE 4. The POLL receiver component.

in response to a POLL PDU. The component (the smallest of all five in our system) is depicted, somewhat simplified for better understanding, in Figure 4. Execution starts in the *DataTransferReady* location by receiving a POLL input, which carries two numerical values: *mVTS* and *mVTPS*. These values respectively denote the values of the sender's highest index of an DATA PDU, and the index of the POLL PDU at the time it was emitted.

The POLL is first checked for validity, which essentially means that its *mVTS* field is not less than *VRH*, the highest index of an DATA PDU that the receiver knows about. If the POLL is not valid, the receiver exits the data transfer mode, which is encoded here by going to the *OutOfDtr* location. An invariant of the system (data not shown) allows us to prove that this location is unreachable. The only stable locations for the POLL receiver component are *DataTransferReady* and *OutOfDtr*.

Then, *VRH* and the receiver's own copy of *VTPS* are updated from the *mVTS* and *mVTPS* fields of the POLL PDU, respectively; and the system starts building a new STAT PDU, to be emitted as a response to the received POLL. The STAT will contain an abstract view of the receiver's window, namely, a list of indices that encodes the empty slots (DATA not arrived, assumed to be lost) and full slots (containing an arrived DATA) in the receiver's window. The rest of the component is essentially the process of building this list *vList*. The result is that *vList* is a strictly increasing sequence of integers $i_1, i_2, \ldots$ with the property:

(ii) *If j is even, then all indices in the interval $[i_{j+1}, i_{j+2})$ denote empty slots in the receiver's window.*

The empty slots in the receiver's window are those in which expected messages did not arrive. Hence, the actual encoding of statement (i) in PVS is given in Figure 5. Thus, we have to prove that the POLL receiver component, depicted in Figure 4, preserves the invariant `retrans_inv1` depicted in Figure 5.

By applying our PVS strategy for proving invariants (cf. Section 2.2) we obtain that, in order to prove the `retrans_inv1` invariant, the auxiliary predicate depicted in Figure 6 should be proved invariant. Note that the latter is just the PVS encoding of the property (ii) above. Our dedicated PVS strategy is applied again, which suggests to prove the two predicates depicted in Figure 7 are invariants. The latter are inductive and are automatically proved by our strategy, which implies that the auxiliary predicate depicted in Figure 6 is an invariant, which in turn implies that the predicate depicted in Figure 5 is an invariant as well.

5.4. Discussion

The verification has only generated auxiliary invariants (cf. Figures 6 and 7) that are *unstable* according to Definition 9 (the only stable locations for the POLL receiver component are *DataTransferReady* and *OutOfDtr*). Unstable invariants are of the form $(\mathrm{pc} = l) \Rightarrow P'$ for an unstable location l, and have the advantage of not requiring a proof of preservation by the other component. That is, we do not need to prove that the sender preserves the predicates depicted in Figures 6 and 7, because, intuitively, while the receiver is in an unstable location, the sender does not move.

```
retrans_inv1: LEMMA invariant(LAMBDA(s:State) :
FORALL (k: subrange(RS_sender_index, s`RS_receiver_index - 1)):
                LET pdu = s`RS_channel(k) IN
                  STAT?(pdu) IMPLIES
                   LET statlist = vList(pdu) IN
                      (FORALL (l: upto(statlist`Length - 2)):
                         even?(l) IMPLIES
                          LET elt1 = statlist`Data(l + 1),
                           elt2 = statlist`Data(l + 2)
                            IN FORALL (m: subrange(elt1,elt2-1)):
                                NOT RecvBuffer`Arrived(m) OR
                               XmitBuffer`VTPS(m) >= mVTPS(pdu)))
```

FIGURE 5. PVS invariant that encodes Property (i).

```
retrans_inv1_aux1: LEMMA invariant(LAMBDA(s:State) :
   s`pc = DtrPollSendList
               IMPLIES
               (FORALL (l: nat):
                  even?(l) AND l <= s`vList`Length - 2 IMPLIES
                   (FORALL (m: subrange(s`vList`Data(l + 1),
                                        s`vList`Data(l + 2) - 1)):
                      NOT RecvBuffer`Arrived(m))))
```

FIGURE 6. Auxiliary invariant for `retrans_inv1`.

```
retrans1_inv1_aux1_aux1: LEMMA
 invariant(LAMBDA (s: State):
            (s`pc = DtrPollScanMissing AND s`vList`Length > 0) IMPLIES
             (FORALL (m: subrange(s`vList`Data(s`vList`Length), s`i - 1)):
                NOT RecvBuffer`Arrived(m)))

retrans1_inv1_aux1_aux2:: LEMMA invariant(LAMBDA(s:State) :
              (s`pc = DtrPollSkipArrived IMPLIES even?(s`vList`Length))
                AND
              (s`pc = DtrPollScanMissing IMPLIES odd?(s`vList`Length)))
```

FIGURE 7. Inductive auxiliary invariants used for proving the invariant in Figure 6.

This saves not only two proofs, but potentially many more, because an invariance proof typically requires a whole *tree* of auxiliary invariants, whose width and depth cannot be anticipated beforehand.

This is the advantage of having employed the partial-order based abstraction. However, the compositional rule (Proposition 8) still requires us to prove that one invariant: the one depicted in Figure 5, is preserved by the sender of the

SSCOP. That branch is treated in a similar manner and is eventually solved as well.

6. CONCLUSION AND RELATED WORK

We describe a methodology based on mechanized compositional and deductive reasoning, and illustrate it by verifying safety properties of the SSCOP protocol. The protocol is decomposed into 'components' that correspond to the transitions of its standard specification [12] expressed in the SDL language.

The verification is compositional in that each component has to ensure only the properties that concern it. The number of proof obligations is kept low thanks to the so-called partial-order abstraction.

The verification is deductive and makes intensive use of the PVS theorem prover. Starting with a set of predicates that constitute initial proof obligations, the user is assisted by PVS in discovering more properties of the protocol, which are indicated by the repeated failed attempts of PVS at proving the protocol correct. The supplementary properties constitute new proof obligations and enrich the user's knowledge of the protocol.

Each PVS proof consists of applying essentially the same automatic strategy, where the user only has to provide adequate quantifier instantiations and, in case the proof fails, to interpret the pending subgoals generated by PVS as new proof obligations. For the current case study, which is the core of a real communication protocol, the verification took 3 months (an additional month was spent for understanding the protocol and translating it to PVS). We believe this is reasonable, as the very definition of the protocol [12] also took several months. If the verification were done in parallel with the definition, this would not significantly increase the duration of the process and may even save time. Typically, the people in charge of proving the protocol acquire a very good understanding of it and can point to errors early in the design phase.

6.1. Related work

We present some of the numerous works related to each of the main techniques (*partial-order-based*, *compositional* and *deductive* verification) employed in this paper. We also mention the *predicate abstraction-refinement* technique, which is currently one of the most successful automatic technique for verifying real software. On the other hand, *communication protocols* have for a long time been a favourite application domain for the verification community; we here list some recent contributions involving relatively large case studies. Finally, we compare the present version of this work with the preliminary version [23].

6.1.1. Partial-order-based verification

Ideas about partial-order abstractions go back as far as [24], where parallel programs executing *communication-closed layers* (similar to our macro-steps) are considered. Equivalence of full interleaved executions and of executions in communication-closed layers is proved using a symmetrical independence relation.

Partial-order-based techniques have been also intensively studied and used by the model-checking community. The basic idea is that, if some transitions are *independent*, then not all their interleavings need to be explored for establishing their properties. This is employed in model checking for combating the state-space explosion problem. The approaches differ on the symmetrical or asymmetric nature of the independence relations employed, on the relative precision of the relation (i.e. which properties are preserved when independent transitions are reordered), and on the manner of incorporating the partial-order reductions in the model-checking algorithms.

Historically, symmetrical independence relations (i.e. if τ_1 is independent of τ_2, then τ_2 is independent of τ_1 as well) have been the first employed. The article [17] studies their fundamental properties. Applications to model checking with on-the-fly detection of independent transitions are presented in the monography [16]. The SPIN model checker [4] uses partial orders intensively, sometimes with dramatic effects on the reduction of the state-space explored. Asymmetric independence relations (τ_1 may be independent of τ_2, but the converse may not necessarily be the case) have been introduced more recently [15].

The independence relation that we employ in this paper (Definition 10) is asymmetric, and the sufficient conditions to establish it (Proposition 5) are undecidable. Hence, the relation is not well adapted to model checking, because, even in the finite-state case, model checking the sufficient conditions may be as complex as model checking the original properties of interest (i.e. the whole state-space of the parallel composition may need to be explored). On the other hand, the independence relation was well suited to the present theorem-proving framework, because establishing it generated acceptable verification overhead (28 invariants automatically proved by our PVS strategy, compared to a total number of 233 invariants for the whole proof). It is expected that our approach works best for loosely coupled systems, for which establishing the independence relation is compensated by the gains resulting from the compositional partial-order verification.

The article [25] describes a theory for combining partial-order abstraction and deductive verification. The theory is deeply embedded in PVS, i.e. PVS is not only used for verifying specific systems, but also meta-level properties such as sufficient criteria for independence of parallel executions. In contrast, we prove these properties by hand and use PVS only to discharge proof obligations for the verification of specific systems. While their approach is more rigorous and involves more creativity with PVS, ours tends to involve the same basic routine with the prover. Compositionality is not an issue in [25].

6.1.2. Compositional verification

The reference monography [26] defines compositionality as the ability not only to verify, but also to build provably correct systems from their specifications. This is indeed a desirable goal. In this view, however, each component may only know the interfaces of the other components, but not their internal details. Hence, according to the above definition of compositionality, our approach is not compositional, and no verification method for proving properties involving internal variables of two different components is compositional either.

Compositionality can also be seen in a broader sense, which can be summarized by the statement 'divide and conquer'. Other authors also adopt this view (e.g. [27] describes compositional verification using the SMV model checker; [28] present a compositional state-space generation for verifying finite-state systems). Another compositional model checking tool is the MOCHA tool [29], which implements the assume-guarantee paradigm in a model-checking framework. Assume-guarantee is truly compositional in the sense of [26].

Among the many verification techniques presented in [26], ours is closest to the classical (also non-compositional in the sense of [26]) method of Owicki and Gries [30] for shared-variable concurrent processes and properties. An important difference is that we provide a systematic method to obtain inductive invariants (which is the hardest part of the verification process). In contrast, the Owicki-Gries method assumes that all invariants provided by the user are inductive right from the start, and does not use abstractions to reduce the number of proof obligations. On the other hand, their method handles more general systems (i.e. that do not satisfy independence hypotheses) and also deal with other classes of properties, e.g. termination. A truly compositional verification method for systems with shared variables is the rely-guarantee approach [31]. Recent compositional approaches based on learning algorithms include [32, 33].

6.1.3. Predicate abstraction-refinement

This is currently one of the most successful techniques for veryfying real software. It originates in the works of Graf and Saïdi [34] and has received sustained academic interest since then. It has been implemented in a variety of tools, including SLAM [2] and BLAST [35], which have been quite successful at automatically verifying C code. This success is demonstrated by the fact that SLAM is the core verification engine in the Static Driver Verifier, part of the Microsoft Windows Driver Foundation, the development framework of device drivers for the omnipresent operating system.

The idea behind predicate abstraction is essentially to discover the relevant information (under the form of predicates on a program's data) that 'governs' the executions of the program. Automatic theorem-proving is used for building an abstract program, whose variables are all Booleans that keep track of the truth values of the chosen predicates. If the abstraction is too coarse, e.g. for proving a given safety property, new predicates (discovered by exploiting information from the counter-example to the safety property, provided by a model checker) are used to build a more precise abstract program. Although the process does not terminate in general, because of undecidability problem, it has been demonstrated to effectively terminate in many cases.

Conceptually, discovering adequate predicates for abstraction and discovering auxiliary inductive invariants (as in this work) is essentially the same thing: both use failures of earlier proof attempts for gaining new, useful information about the system. However, the automatic heuristics for abstraction refinement can only discover relatively simple predicates, unlike the complex invariants required for the present case study. It is reasonable to assume that really complex invariants can only be inferred by the user of a theorem prover, using both feedback from the prover *and* knowledge gradually gained while verifying the case study.

6.1.4. Verification of communication protocols

Relevant works in the deductive verification of communication protocols include [19, 36, 37, 38, 39, 40]. Deductive verification as a stand-alone technique is perceived, with good reason, as being time-consuming and requiring a lot of user expertise. It is, however, useful in combination with abstraction and model-checking techniques. The history of Phillips's Bounded Retransmission Protocol (a protocol fairly simpler than the SSCOP) is revealing for that matter: the initial deductive verification of the protocol [39] was performed in COQ [5] and took 3 months. The deductive verification of the same protocol with the more automated PVS system took 1 month [19]. Using the most recent features of abstraction and model checking in PVS [18] the protocol was automatically verified in a matter of seconds.

The automatic verification of protocols specified in SDL has also received some attention recently [41, 42, 43, 44]. These approaches are based on model checking, thus, they are subject to the usual limitations: some finite, usually 'small' instances (in terms of the size of buffers and communication channels) of 'large' case studies are verified. On the other hand, all these model-checking based techniques are fully automatic, unlike the combination of techniques presented in this paper, which rely on semi-automatic theorem proving. Compositionality and abstraction are also used to reduce a large state space to one amenable to model checking, but the corresponding reduction rules are not always formally justified. In [21], the SSCOP protocol is model-checked for safety and liveness properties, under more restrictive

hypotheses than ours: perfect FIFO channels with a small fixed size.

Finally, in a preliminary version of this paper [23], we have verified the SSCOP protocol based on a stronger, *syntactical* definition of atomic sequences:

- In [23] an atomic sequence of transitions may access a shared variable (here, any of the communication channels) at most once, either in reading or in writing. Moreover, if the access is a reading it must occur at the beginning, and if it is a reading, it must occur at the end of the sequence.
- In the present paper, this definition is semantical—but for the case of communication channels, the definition amounts to the fact that an atomic sequence may (optionally) start with one input from one channel, followed by an arbitrary sequence of internal actions and outputs to the other channel.

The definition of atomic sequences in the present paper matches the notion of transition in the SDL language. Hence, in this paper we verify the SSCOP protocol in its precise SDL semantics, whereas in [23], an over-aproximation of the semantics was used. Although over-approximation is sound for the class of safety properties, it did lead the proof into useless branches that were quite hard to comprehend because they refer to executions that do not exist to the actual SDL semantics.

REFERENCES

[1] Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A. (2002) NuSMV 2: In *Proc. 14th Into. Conf. Computer Aided Verification (CAV'02)*, Copenhagen, Denmark, July 27–31, pp. 359–364. Springer-Verlag, Berlin.

[2] Lahiri, S. K., Ball, T. and Cook, B. (2005) Predicate abstraction via symbolic decision procedures. In *Proc. 17th Int. Conf. Computer Aided Verification (CAV'05)*, Edinburgh, Scotland, July 6–10, pp. 24–38. Springer-Verlag, Berlin.

[3] Garavel, H., Lang, F. and Mateescu, R. (2001) *An overview of CADP 2001*. Technical Report RT-0254. INRIA, Grenoble, France.

[4] Holzmann, G. (2003) *The Spin Model Checker: Primer and Reference Manual*, Addison-Wesley, New York.

[5] Bertot, Y. and Casteran, P. (2004) *Coq'Art: The Calculus of Inductive Constructions*, Springer-Verlag, Berlin.

[6] Gordon, M. and Melham, T. (1993) *Introduction to HOL: A Theorem Proving Environment for Higher-Oder Logic*, Cambridge University Press, Cambridge, UK.

[7] Nipkow, T., Paulson, L. C. and Wenzel, M. (2002) *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, Springer Verlag, Berlin.

[8] Owre, S., Rushby, J. M., Shankar, N. and von Henke, F. W. (1995) Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. Software Eng.*, **21**, 107–125.

[9] Pnueli, A. (2000) Keynote address: abstraction, composition, symmetry, and a little deduction: the remedies to state explosion. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*, Chicago, IL, USA, July 15–19, pp. 1–2. Springer-Verlag, Berlin.

[10] Bensalem, S., Lakhnech, Y. and Owre, S. (1998) Computing abstractions of infinite state systems compositionally and automatically. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98)*, Vancouver, BC, Canada, June 28–July 2, pp. 319–331. Springer-Verlag, Berlin.

[11] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L. and Hwang, L. J. (1992) Symbolic model checking: 10^{20} states and beyond. *Inform. Comput.*, **98**, 142–170.

[12] Recommendation Q.2110 (1994) *ATM Adaptation Layer—Service Specific Connection Oriented Protocol*.

[13] Henderson, T. and Katz, R. (1999) Transport protocols for internet-compatible satellite networks. *IEEE J. Selected Areas Communi.*, **17**, 326–244.

[14] Telelogic SDL products http://www.telelogic.com/products/sdl.

[15] Abdulla, P. A., Jonsson, B., Kindahl, M. and Peled, D. (1998) A general approach to partial order reductions in symbolic verification (extended abstract). In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98)*, Vancouver, BC, Canada, June 28–July 2, pp. 379–390. Springer-Verlag, Berlin.

[16] Godefroid, P. (1996) *Partial-Order Methods for the Verification of Concurrent Systems—An Approach to the State-Explosion Problem*, Springer Verlag, Berlin.

[17] Katz, S. and Peled, D. (1992) Defining conditional independence using collapses. *Theor. Comp. Sci.*, **101**, 337–359.

[18] Saïdi, H. and Shankar, N. (1999) Abstract and model check while you prove. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99)*, Trento, Italy, July 6–10, pp. 443–454. Springer-Verlag, Berlin.

[19] Havelund, K. and Shankar, N. (1996) Experiments in theorem proving and model checking for protocol verification. In *Proc. Third Int. Symp. Formal Methods Europe (FME'96)*, Oxford, UK, March 18–22, pp. 662–681. Springer-Verlag, Berlin.

[20] Bozga, M., Fernandez, J.-C., Ghirvu, L., Graf, S., Krimm, J.-P. and Mounier, L. (2000) If: a validation environment for timed asynchronous systems. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*, Chicago, IL, USA, July 15–19, pp. 543–547. Springer-Verlag, Berlin.

[21] Bozga, M., Fernandez, J.-C., Ghirvu, L., Jard, C., Jéron, T., Kerbrat, A., Morel, P. and Mounier, L. (2000) Verification and test generation for the sscop protocol. *Science of Computer Programming*, **36**, 27–52.

[22] Rusu, V. (2004) *Verifying an ATM Protocol Using A Combination of Formal Techniques*, Technical Report 5089. INRIA.

[23] Rusu, V. (2003) Compositional verification of an ATM protocol. In *Proc. Int. Sym. Formal Methods Europe (FME'03)*, Pisa, Italy, September 8–14, pp. 223–243. Springer-Verlag, Berlin.

[24] Elrad, T. and Francez, N. (1982) Decomposition of distributed programs into communication-closed layers. *Sci. Comput. Program.*, **2**, 155–173.

[25] Glusman, M. and Katz, S. (1999) Mechanizing proofs of computation equivalence. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99)*, Trento, Italy, July 6–10, pp. 354–367. Springer-Verlag, Berlin.

[26] de Roever, W.-P., de Boer, F., Hannemann, U., Hooman, J, Lakhnech, Y., Poel, M. and Zwiers, J. (2001) *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, Cambridge, UK.

[27] McMillan, K. L. (1997) A compositional rule for hardware design refinement. In *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97)*, Haifa, Israel, June 22–25, pp. 24–35. Springer-Verlag, Berlin.

[28] Krimm, J.-P. and Mounier, L. (2000) Compositional state space generation with partial order reductions for asynchronous communicating systems. In *Proc. 6th Int. Conf. TATools and Algorithms for Construction and Analysis of Systems (CAS'00)*, Berlin, Germany, March 25–April 2, pp. 266–282. Springer-Verlag, Berlin.

[29] Alur, R., Henzinger, T. A., Mang, F. Y. C., Qadeer, S., Rajamani, S. K. and Tasiran, S. (1998) Mocha: modularity in model checking. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98)*, Vancouver, BC, Canada, June 28–July 2, pp. 521–525. Springer-Verlag, Berlin.

[30] Owicki, S. S. and Gries, D. (1976) An axiomatic proof technique for parallel programs. *Acta Inform.*, **6**, 319–340.

[31] Jones, C. (1983) Tentative steps towards a development method for interfering programs. *ACM Trans. Programming Lang. Syst.*, **5**, 596–619.

[32] Alur, R., Madhusudan, P. and Nam, W. (2005) Symbolic compositional verification by learning assumptions. In *Proc. 17th Int. Conf. Computer Aided Verification (CAV'05)*, Edinburgh, Scotland, July 6–10, pp. 548–562. Springer-Verlag, Berlin.

[33] Chaki, S., Clarke, E. M., Sinha, N. and Thati, P. (2005) Automated assume-guarantee reasoning for simulation conformance. In *Proc. 17th Int. Conf. Computer Aided Verification (CAV'05)*, Edinburgh, Scotland, July 6–10, pp. 534–547. Springer-Verlag, Berlin.

[34] Graf, S. and Saïdi, H. (1997) Construction of abstract state graphs with PVS. In *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97)*, Haifa, Israel, June 22–25, pp. 72–83. Springer-Verlag, Berlin.

[35] Henzinger, T. A., Jhala, R., Majumdar, R. and Sutre, G. (2003) Software verification with BLAST. In *Proc. Model Checking Software, 10th Int. SPIN Workshop (SPIN'03)*, Portland, OR, May 9–10, pp. 235–239. Springer-Verlag, Berlin.

[36] Bickford, M., Kreitz, C., van Renesse, R. and 0003, X. L. (2001) Proving hybrid protocols correct. In *Proc. 14th Int. Conf. Theorem Proving in Higher Order Logics (TPOLs'01)*, Edinburgh, Scotland, UK, September 3–6, pp. 105–120. Springer-Verlag, Berlin.

[37] Cardell-Oliver, R. (1991) On the use of the HOL system for protocol verification. In *Proc. Int. Workshop on the HOL Theorem Proving System and its Applications (TPHOLs'91)*- Davis, California, USA, August, pp. 59–62. IEEE Computer Society.

[38] Chkliaev, D., Hooman, J. and de Vink, E. P. (2003) Verification and improvement of the sliding window protocol. In *Proc. 9th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, Warsaw, Poland, April 7–11, pp. 113–127. Springer-Verlag, Berlin.

[39] Helmink, L., Sellink, M. P. A. and Vaandrager, F. W. (1993) Proof-checking a data link protocol. In *Proc. Int. Workshop, Selected Papers Types for Proofs and Programs (TYPES'93)*, Nijmegen, The Netherlands, May 24–28, pp. 127–165. Springer-Verlag, Berlin.

[40] Nipkow, T. and Slind, K. (1994) I/Q automata in Isabelle/HOL. In *Proc. Int Workshop, Selected Papers Types for Proofs and Programs (TYPES'94)*, Båstad, Sweden, June 6–10, pp. 101–119. Springer-Verlag, Berlin.

[41] Bosnacki, D., Dams, D., Holenderski, L. and Sidorova, N. (2000) Model checking sdl with spin. In *Proc. 6th Int. Conf. Tools and Algorithms for Construction and Analysis of Systems (TACAS'00)*, Berlin, Germany, March 25– April 2, pp. 363–377. Springer-Verlag, Berlin.

[42] Jia, G. and Graf, S. (2001) Verification experiments on the MASCARA protocol. In *Proc. 8th Int. SPIN Workshop Model Checking Software (SPIN'01)*, Toronto, Canada, May 19–20, pp. 123–142. Springer-Verlag, Berlin.

[43] Ioustinova, N., Sidorova, N. and Steffen, M. (2002) Closing open sdl-systems for model checking with DTSpin. In *Formal Methods, International Symposium of Formal Methods Europe (FME'02)*, Copenhagen, Denmark, July 22–24, pp. 531–548. Springer-Verlag, Berlin.

[44] Sidorova, N. and Steffen, M. (2001) Verifying large sdl specifications using model checking. In *Proc. 10th International SDL Forum (SDL'01)*, Copenhagen, Denmark, June 27–29, pp. 402–420. Springer-Verlag, Berlin.

APPENDIX

PROPOSITION 1. *Let* $\sigma : l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ *be a finite, contiguous sequence of transitions of an extended automaton* $\mathcal{E}$. *Let* ρ_i *denote the transition relation of the transition* τ_i, *for* $i = 1, \ldots, n$, *and* ρ *denote the transition relation of the transition* $\tau = Collapse(\sigma)$. *Then (†):* $\rho = \rho_n \bullet \rho_{n-1} \bullet \cdots \bullet \rho_1$.

Proof. By induction on the length n of the sequence σ. The base step ($n = 1$) is trivial. For the induction step, assume that the equality (†) in the statement of Proposition 2.3 holds for sequences of length $n - 1$, in particular for the sequence $\sigma' : l_1 \xrightarrow{\tau_1} l_2 \cdots \xrightarrow{\tau_{n-1}} l_n$.

Hence, if ρ' is the transition relation of the transition $\tau' = Collapse(\sigma')$, then the equality (‡): $\rho' = \rho_{n-1} \bullet \cdots \bullet \rho_1$ holds. By construction, the guard of τ' is $G' = G_1 \wedge (G_2 \circ A_1) \wedge \cdots (G_{n-1} \circ A_{n-2} \circ \cdots \circ A_1)$, and its assignments are $A' = A_{n-1} \circ A_{n-2} \circ \cdots \circ A_1$. Then, the guard of $\tau = Collapse(\sigma)$ can be written as $G = G' \wedge G_n \circ \mathrm{A}'$, and its assignments as $A = A_n \circ A'$. We now prove (§): $\rho = \rho_n \bullet \rho'$.

($\subseteq$) Let (s, s') be an arbitrary pair of states in ρ. Then, by Definition 5 of the *Collapse* operation, and using Definition 4 of the transition relation of a transition:

(i) $s = \langle l_1, v\rangle$ for some valuation v of the variables V such that v satisfies $G = G' \wedge G_n \circ A'$,
(ii) $s' = \langle l_{n+1}, v'\rangle$ for the valuation v' such that $v' = A(v) = (A_n \circ A')(v) = A_n\,(A'(v))$,

where l_1 is the origin of the transition τ_1 and l_{n+1} is the destination of the transition τ_n.

Let s'' be the state $\langle l_n, v''\rangle$ with $v'' = A'(v)$, where l_n is the destination of τ_{n-1}. By Definition 5, l_1 is also the origin, and l_n is the destination of $\tau' = \mathit{Collapse}(\sigma')$. Then,

- $\underline{(s, s'') \in \rho'}$, as the location l_1 of s is the origin of τ'; the valuation v of s satisfies G' [cf. item (i)]; the location l_n of s'' is the destination of τ'; and the valuation of s'' is $A'(v)$,
- $\underline{(s'', s') \in \rho_n}$, as the location l_n of s'' is the origin of τ_n (as the sequence σ is contiguous); the valuation v'' of s'' does satisfy G_n [we have seen—cf. item (i) above—that v satisfies $G_n \circ A'$, hence, $v'' = A'(v)$ satisfies G_n]; the location l_{n+1} of s' is the destination of τ_n; and the valuation v' of s' satisfies $v' = A(v) = (A_n \circ A')(v) = A_n\,(A'(v)) = A_n\,(v'')$.

The underlined items and Definition 6 imply $(s, s') \in \rho_n \bullet \rho'$, and the $\subseteq$ inclusion is proved.

($\supseteq$) If $(s, s') \in \rho_n \bullet \rho'$, then, by Definition 6, there exists $s'' \in S$ such that $(s, s'') \in \rho'$, and $(s'', s') \in \rho_n$. From $(s, s'') \in \rho'$ and $(s'', s') \in \rho_n$ we obtain

(i) $s = \langle l_1, v\rangle$, for some valuation v of the variables V satisfying G',
(ii) $s'' = \langle l_n, v''\rangle$ with $v'' = A'(v)$,
(iii) $v'' = A'(v)$ satisfies G_n, hence, v satisfies $G_n \circ A'$; as v also satisfies G'; (cf. (i) above) we obtain that v satisfies $G' \wedge G_n \circ A'$. As $G = G' \wedge G_n \circ A'$, $\underline{v \text{ satisfies } G}$,
(iv) $s' = \langle l_{n+1}, v'\rangle$ for the valuation $v' = A_n\,(v'') = (A_n \circ A')(v)$. As $A = A_n \circ A'$, $\underline{v' = A(v)}$.

Hence, $s = \langle l_I, v\rangle$ is such that its location l_1 is the origin of the transition $\tau = \mathit{Collapse}(\sigma)$, and its the valuation v satisfies the guard G of τ; and $s' = \langle l_{n+1}, v'\rangle$ is such that its location l_{n+1} is the destination of the transition τ, and its valuation $v' = A(v)$ is obtained by applying the assignments A of τ to v. We then conclude that (s, s') belongs to ρ, the transition relation of τ, and the equality (§) is proved.

Finally, from (§) and the induction hypothesis (‡) we obtain $\rho = \rho_n \bullet \rho_{n-1} \bullet \cdots \bullet \rho_1$, i.e. the equality (†) in Proposition 1, and the proof is done.

PROPOSITION 2. *An extended automaton has finitely many maximal syntactical macro-steps.*

Proof. By Definition 7, a maximal syntactical macro-step σ does not contain cycles of unstable locations. Thus, either σ does not contain cycles at all (and there are finitely many such *acyclic* sequences of transitions in any given graph), or σ contain a cycle that does include a *stable* location. Now, by Definition 1, the only stable locations on σ are its first and last location. Hence, if σ does contain a cycle with a stable location on it, then σ itself is reduced to an *elementary* cycle (which that starts and ends in the same location), of which there are only finitely many in any given graph.

PROPOSITION 3. *Let $\mathcal{E}$ be an extended automaton and $\mathit{Collapse}(\mathcal{E})$ its collapsed automaton. Then, a stable state is reachable in $\mathcal{E}$ if and only if it is also reachable in $\mathit{Collapse}(\mathcal{E})$.*

Proof. ($\Rightarrow$) Assume that s is a stable state (i.e. s is of the form $\langle l, v\rangle$ where l is a stable location of $\mathcal{E}$) that is reachable in $\mathcal{E}$. We prove that s is also reachable in $\mathit{Collapse}(\mathcal{E})$. By definition, s is reachable if it is the last state of a run r of $\mathcal{E}$. The proof is done by well-founded induction on the run r, where runs are ordered by the (strict) prefix relation.

For the base step, it is enough to note that the initial states of $\mathcal{E}$ and of Collapse($\mathcal{E}$) are the same. This is because the initial conditions of the two extended automata are the same, and so are the initial locations (the *Collapse* operation may at most remove *unstable* locations from an automaton, which, by definition, do not include the initial locations).

For the induction step, assume that the stable state s of $\mathcal{E}$ is not initial, and is the last state of some run r. Hence, there exists a strict prefix r' of the run r, such that:

- the last state s_1 of r' is stable;
- there exists a run fragment $r'' : s_1, \ldots, s_n = s$ between s_1 and s, which is a (semantical) macro-step.

Let $s_i = \langle l_i, v_i\rangle$, for $i = 1, \ldots, n$. By Definition 4 of a run fragment, there exists a sequence of transitions $\tau_1, \ldots \tau_{n-1}$ such that, for $i = i, \ldots, n-1$, $(s_i, s_{i+1}) \in \rho_i$, where ρ_i denotes the transition relation of transition τ_i; and, using again Definition 4 (semantical macro-step), the sequence $\sigma : l_1 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_{n-1}} l_n$ is a *syntactical* macro step, which is either of the following:

(i) A maximal syntactical macro-step. In this case, by Definition 8, the automaton *Collapse* ($\mathcal{E}$) contains the transition $\tau = \mathit{Collapse}(\sigma)$ which, by Proposition 1, has the transition relation $\rho = \rho_{n-1} \bullet \rho_{n-2} \bullet \cdots \bullet \rho_1$. Hence, using Definition 6 of the composition of binary relations, we obtain $(s_1, s) \in \rho$.
(ii) A non-maximal syntactical macro step. In this case, by Definition 8, the automaton Collapse ($\mathcal{E}$) contains the whole syntactical macro-step σ, which means that the semantics of $\mathit{Collapse}(\mathcal{E})$ admits the semantical macro-step r'': $s_1, \ldots, s_n = s$.

By induction hypothesis, s_1 is also reachable in $Collapse(\mathcal{E})$, that is, there exists a run $\tilde{r}$ of $Collapse(\mathcal{E})$ such that s_1 is the last state of $\tilde{s}$. In Case (i) above, the definition of runs ensures that $\tilde{r} \cdot s$ is a run of $Collapse(\mathcal{E})$, i.e. s is reachable in $Collapse(\mathcal{E})$; but in Case (ii) the run $\tilde{r} \cdot r''$ shows that s is reachable in $Collapse(\mathcal{E})$ as well: the ($\Rightarrow$) implication is proved.

($\Leftarrow$) The proof of this implication is similar to the previous one, it is here only sketched. If s is a stable state of $Collapse(\mathcal{E})$, then either s is an initial state of $Collapse(\mathcal{E})$, and we have seen that the initial states of $\mathcal{E}$ and of $Collapse(\mathcal{E})$ are the same; or s is obtained by firing a sequence of transitions that constitutes a syntactical macro-step σ of $Collapse(\mathcal{E})$, starting from another stable state s_1 that is reachable in $Collapse(\mathcal{E})$ (and also in $\mathcal{E}$, by the induction hypothesis). By Definition 8, σ is either reduced to one transition $Collapse(\sigma')$, or $\sigma = \sigma'$, for some syntactical macro-step σ' of $\mathcal{E}$. In both cases, we prove [just as in the proof of the ($\Rightarrow$) implication) that the effects of executing the sequence σ in $Collapse(\mathcal{E})$, and σ' in $\mathcal{E}$, are the same. Hence, since s was reached from s_1 in $Collapse(\mathcal{E})$ by firing the sequence σ, then s is reached from s_1 by firing the sequence σ' in $\mathcal{E}$. By the induction hypothesis, the state s_1 it reachable in $\mathcal{E}$, and we obtain that s is reachable in $\mathcal{E}$.

PROPOSITION 4. *If P is a stable predicate of $\mathcal{E}$, then $Collapse(\mathcal{E}) \models \Box P$ iff $\mathcal{E} \models \Box P$.*

Proof. We prove that $Collapse(\mathcal{E}) \not\models \Box P$ iff $\mathcal{E} \not\models \Box P$, and the proposition follows. We have (1): $Collapse(\mathcal{E}) \not\models \Box P$ iff (2): there exists a state s that is reachable in $Collapse(\mathcal{E})$ and violates P. Now, P is stable, i.e. it is of the form $(pc = l) \Rightarrow Q$, for l a stable location of $\mathcal{E}$. The fact that s violates P, is equivalent to s satisfies $\neg P : pc = l) \wedge \neg Q$, i.e. s has the form $s = \langle l, v \rangle$, for l a stable location of $\mathcal{E}$, i.e. s is stable. Thus, (2) is equivalent to (3): there exists a *stable* state s that is reachable in $Collapse(\mathcal{E})$ and violates P. By Proposition 2 above, (3) is then equivalent to (4): there exists a *stable* state s that is reachable in $\mathcal{E}$ and violates P; since P is stable, (4) is then equivalent to $\mathcal{E} \not\models \Box P$, and the proof is done.

PROPOSITION 5. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be extended automata, τ_1 a transition of $\mathcal{E}_1$, and τ_2 a transition of $\mathcal{E}_2$. Let G_1 denote the guard of τ_1, l_1 denote the origin of τ_1, and A_1, A_2 denote respectively the assignments of τ_1, τ_2. Then, τ_2 is independent of τ_1 in every reachable state of $\mathcal{E}_1 \| \mathcal{E}_2$ if*

(i) $\mathcal{E}_1 \| \mathcal{E}_2 \models \Box[(pc_1 = l_1 \wedge G_1) \Rightarrow pre_{\tau_2}(G_1)]$ *and*
(ii) $\mathcal{E}_1 \| \mathcal{E}_2 \models \Box[(pc_1 = l_1 \wedge G_1) \Rightarrow rhs(A_1 \circ A_2) = rhs(A_2 \circ A_1)]$

Proof. We have to prove that, if conditions (i) and (ii) above hold, then, for all reachable states s and states s', s'', if $s \xrightarrow{\tau_1} s' \xrightarrow{\tau_2} s''$ then there exists a state $\tilde{s}$ such that $s \xrightarrow{\tau_2} \tilde{s} \xrightarrow{\tau_1} s''$. From the fact that τ_1 can be fired in s (i.e. from $s \xrightarrow{\tau_1} s'$) we obtain that the state s satisfies $(pc_1 = l_1) \wedge G_1$. As s is reachable in $\mathcal{E}_1 \| \mathcal{E}_2$, we obtain from the hypothesis (i) in the statement of the proposition that s satisfies the predicate $pre_{\tau_2}(G_1)$.

By the definition of *pre* [Equation (2), Page 20] this means that there exists a state $\tilde{s}$ such that $s \xrightarrow{\tau_2} \tilde{s}$ and that G_1 is true in $\tilde{s}$. Then, τ_1 is fireable in $\tilde{s}$, because its guard G_1 is true, and $pc_1 = l_1$ still holds after the firing of τ_2 (the firing of τ_2 by $\mathcal{E}_2$ has no effect on the control pc_1 of $\mathcal{E}_1$). Let then $\hat{s}$ be a state such that $\tilde{s} \xrightarrow{\tau_1} \hat{s}$. To complete the proof we only need to show that $\hat{s} = s''$. For this, it is enough to note that hypothesis (ii) in the statement of the proposition ensures that in state s, the assignments A_1 and A_2 can be executed in any order, yielding the same global result.

PROPOSITION 6. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be mutually independent extended automata, and s a reachable, stable state of $\mathcal{E}_1 \| \mathcal{E}_2$. Then, s is reachable in $\mathcal{E}_1 | \mathcal{E}_2$.*

Proof. Let r be a run of $\mathcal{E}_1 \| \mathcal{E}_2$ that terminates the stable state s. We prove that there exists a run $\tilde{r}$ of $\mathcal{E}_1 | \mathcal{E}_2$ that terminates in the state s as well. The proof is done by well-founded induction on the length of the run r.

The base step is trivial, as the initial states of $\mathcal{E}_1 | \mathcal{E}_2$ and $\mathcal{E}_1 \| \mathcal{E}_2$ are the same and are stable.

For the induction step, assume that the length of the run r is $n \geq 1$, and let s^1 be a predecessor of s on the run r such that (i) the projection s^1_1 of s^1 on $\mathcal{E}_1$ is stable, and (ii) there is no other state on r between s^1 and s whose projection on $\mathcal{E}_1$ is stable.

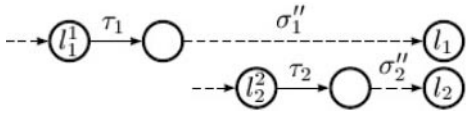
If no *strict* predecessor s^1 of s on r satisfying the above conditions can be found, this means that $\mathcal{E}_1$ has not made any move from the beginning (it has remained in its initial state). Thus, only $\mathcal{E}_2$ has moved; this particular run of $\mathcal{E}_1 \| \mathcal{E}_2$ is clearly also a run $\mathcal{E}_1 | \mathcal{E}_2$ as well, and the proof is done in this case.

Hence, we can assume that s^1 is a strict predecessor of s on the run r. Let s^2 be a state of $\mathcal{E}_1 \| \mathcal{E}_2$ defined similarly to s^1 by considering the component $\mathcal{E}_2$ instead of $\mathcal{E}_1$. Using the same arguments, we can assume that $\mathcal{S}^2$ is a strict predecessor of s on r. Let s^2_2 be the projection of s^2 on $\mathcal{E}_2$; then, by construction, s^2_2 is a stable state of $\mathcal{E}_2$. Clearly, s^1_1, s^2_2 exist and are uniquely defined. Let also s^1, s^2 denote the projections of s on $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively.

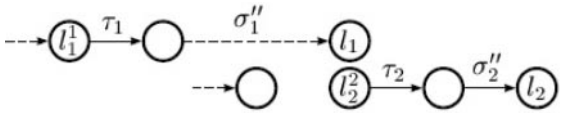
Let l_1, l_2, l^1_1, l^2_2 be the respective locations of the states s_1, s_2, s^1_1, s^2_2. Then, $\mathcal{E}_1$ goes from state s^1_1 to s^1 by taking a sequence of transitions between locations l^1_1 and l_1, which constitutes by construction a non-empty syntactical macro-step σ_1 of $\mathcal{E}_1$; and similarly, there is a non-empty syntactical macro-step σ_2 between locations l^2_2 and l_2 of $\mathcal{E}_2$. The run r terminates once all the transitions of σ_1 and of σ_2 have been fired.

Without restricting the generality we assume that σ_1 is started first, i.e. $\mathcal{E}_1$ fires the first transition τ_1 of $\sigma_1 = \tau_1 \cdot \sigma''_1$

before $\mathcal{E}_2$ fires the first transition τ_2 of $\sigma_2 = \tau_2 \cdot \sigma_2''$. This situation is depicted as follows:



By Definition 11, all the transitions of σ_1'' are independent of all the the transitions of $\mathcal{E}_2$, in particular, of all the transitions in $\sigma_2 = \tau_2 \cdot \sigma_2''$. Then, by Definition 10 (independence of transitions) it is possible to keep the same interleaving up to the beginning of σ_2, and to ''move'' all σ_2 *after* σ_1'', which yields the same global state:



Now, the run r' of $\mathcal{E}_1 \| \mathcal{E}_2$ that terminates after the sequence σ_1'' (see above) is strictly shorter than r, and terminates in a stable state of $\mathcal{E}_1 \| \mathcal{E}_2$, thus, by the induction hypothesis, it can be simulated in $\mathcal{E}_1 | E_2$ by a run r''. Then, the run $\tilde{r}$ obtained by executing $\sigma_2 = \tau_2 \cdot \sigma_2''$ after r'' satisfies all the conditions of the proposition.

PROPOSITION 7. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be mutually independent extended automata and P a stable predicate of $\mathcal{E}_1 \| \mathcal{E}_2$. Then $\mathcal{E}_1 | \mathcal{E}_2 \models \Box P$ holds if and only if $\mathcal{E}_1 \| \mathcal{E}_2 \models \Box P$ holds.*

Proof. ($\Rightarrow$) Let $P : (pc_1 = l_1 \wedge pc_2 = l_2) \Rightarrow P'$. If $\mathcal{E}_1 \| \mathcal{E}_2 \models \Box P$ does not hold, then, there exists a reachable state s of $\mathcal{E}_1 \| \mathcal{E}_2$ in which the predicate $((pc_1 = l_1 \wedge pc_2 = l_2) \Rightarrow P')$ is violated, i.e. $(pc_1 = l_1 \wedge pc_2 = l_2 \wedge \neg P')$ holds in s. Hence, s is stable and by Proposition 2 it is also reachable in $\mathcal{E}_1 | \mathcal{E}_2$, which contradicts $\mathcal{E}_1 | \mathcal{E}_2 \models \Box P$.

($\Leftarrow$) This direction is trivial, as every reachable state in $\mathcal{E}_1 | \mathcal{E}_2$ is also reachable in $\mathcal{E}_1 \| \mathcal{E}_2$

PROPOSITION 8. *Let $\mathcal{E}_1$, $\mathcal{E}_2$* be extended automata over a common set of variable V and P a predicate on the states of $\mathcal{E}_1 | \mathcal{E}_2$ such that P holds in the initial states of $\mathcal{E}_1$, $\mathcal{E}_2$. Let L_s^1, L_s^2 denote the sets of stable locations of $\mathcal{E}_1, \mathcal{E}_2$, respectively.

If $\mathcal{E}_1 \, \mathbf{init} \langle P, L_s^1 \rangle \models \Box P$ and $\mathcal{E}_2 \, \mathbf{init} \langle P, L_s^2 \rangle \models \Box P$ hold, then $\mathcal{E}_1 | \mathcal{E}_2 \models \Box P$ holds.

Proof. By induction on the length of the runs of $\mathcal{E}_1 | \mathcal{E}_2$. For the base step we just show that P holds initially, which is true by construction of $\mathcal{E}_1 | \mathcal{E}_2$ and by the fact that P holds in the initial states of both $\mathcal{E}_1$ and $\mathcal{E}_2$.

For the induction step: assume that P holds in all states of all runs r of $\mathcal{E}_1 | \mathcal{E}_2$ of length n. Now, the run r leaves both $\mathcal{E}_1$ and $\mathcal{E}_2$ in a *stable* location, which, by Definition 13, means that both $\mathcal{E}_1 \, \mathbf{init} \langle P, L_s^1 \rangle$ and $\mathcal{E}_2 \, \mathbf{init} \langle P, L_s^2 \rangle$ are in an *initial* location after r. Moreover, by the induction hypothesis, P holds in the last state of r, thus, after r, $\mathcal{E}_1 \, \mathbf{init} \langle P, L_s^1 \rangle$ and $\mathcal{E}_2 \, \mathbf{init} \langle P, L_s^2 \rangle$ are in an initial *state*. To obtain a run of length $n+1$, either $\mathcal{E}_1$ or $\mathcal{E}_2$ have to perform a macro-step r'. But this run fragment is an actual *run* of either $\mathcal{E}_1 \, \mathbf{init} \langle P, L_s^1 \rangle$ or $\mathcal{E}_2 \, \mathbf{init} \langle P, L_s^2 \rangle$, as we have seen that r' starts in an initial state of both systems.

Without restricting the generality we assume that r' is a run of $\mathcal{E}_1 \, \mathbf{init} \langle P, L_s^1 \rangle$. By $\mathcal{E}_1 \, \mathbf{init} \langle P, L_s^1 \rangle \models \Box P$, we obtain that P still holds in the last state of r' (and of $r \cdot r'$). This concludes the induction step and the proof. □

PROPOSITION 9. *For P a stable state predicate of $\mathcal{E}_1 | \mathcal{E}_2$, $Collapse(\mathcal{E}_1 | \mathcal{E}_2) \models \Box P$ if and only if $Collapse(\mathcal{E}_1) | Collapse(\mathcal{E}_2) \models \Box P$.*

Proof. It is enough to prove that each stable reachable state of $Collapse(\mathcal{E}_1 | \mathcal{E}_2)$ is a stable reachable state of $Collapse(\mathcal{E}_1) | Collapse(\mathcal{E}_2)$. By Proposition 3, this is equivalent to proving (†) that each stable reachable state of $\mathcal{E}_1 | \mathcal{E}_2$ is a stable reachable state of $Collapse(\mathcal{E}_1) | \ Collapse(\mathcal{E}_2)$. We prove (†) and the proposition follows.

($\Rightarrow$) Assume that s is a stable state that is reachable in $\mathcal{E}_1 | \mathcal{E}_2$. We prove that s is also reachable in $Collapse(\mathcal{E}_1) | Collapse(\mathcal{E}_2)$. By definition, s is reachable if it is the last state of a run r of $\mathcal{E}_1 | \mathcal{E}_2$. The proof is done by well-founded induction on the run r, where runs are ordered by the (strict) prefix relation.

For the base step, it is enough to note that the initial states of $Collapse(\mathcal{E}_1) | Collapse(\mathcal{E}_2)$ and of $\mathcal{E}_1 | \mathcal{E}_2$ are the same. This is because the initial conditions of the two extended automata are the same, and so are the initial locations (the *Collapse* operation may at most remove *unstable* locations from an automaton, which do not include the initial locations).

For the induction step, assume that the stable state s of $\mathcal{E}_1 | \mathcal{E}_2$ is not initial, and is the last state of some run r. Hence, there exists a strict prefix r' of the run r, such that:

- the last state s_1 of r' is stable;
- there exists a run fragment r'' $s_1, \ldots, s_n = s$ between s_1 and s, which is a semantical macro-step of $\mathcal{E}_1 | \mathcal{E}_2$.

By the induction hypothesis, s_1 is also reachable in $Collapse(\mathcal{E}_1) | Collapse(\mathcal{E}_2)$. By definition of the atomic parallel composition, we obtain that either $\mathcal{E}_1$ or $\mathcal{E}_2$ move, by performing a semantical macro-step between s_1 and s, while the other one does not move. Now, $s = (s^1, s^2)$, for some stable states s_1 of $\mathcal{E}_1$ and s_2 of $\mathcal{E}_2$, and $s_1 = (s_1^1, s_1^2)$ for some stable states s_1^1 of $\mathcal{E}_1$ and s_1^2 of $\mathcal{E}_2$. Without restricting the generality, we assume that $\mathcal{E}_1$ moves, and $\mathcal{E}_2$ stays still. Thus, $s^2 = s_1^2$ and there exists a semantical macro-step r^1 of

$\mathcal{E}_1$ between s_1^1 and s^1. This semantical macro-step corresponds to a *syntactical* macro-step σ in $\mathcal{E}_1$. Then,

(i) if σ is maximal, then, by Definition 3 there exists in $Collapse(\mathcal{E}_1)$, a transition $\tau = Collapse(\sigma)$ that has the same effect as σ; in particular, that s^1 is reachable from s_1^1 by firing the transition τ. Now, firing this transition constitutes a semantical macro-step of $Collapse(\mathcal{E}_1)$, therefore, by definition of the atomic parallel composition, the state (s^1, s_1^2) is reachable in $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$. As $s^2 = s_1^2$, we obtain that $s = (s^1, s^2)$ is reachable in $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$, and ($\Rightarrow$) is proved in this case.

(ii) if σ is *not* maximal, then by Definition 3, the sequence σ appears in $Collapse(\mathcal{E}_1)$. Then, the semantical macro-step r^1 of $\mathcal{E}_1$ that corresponds to firing the sequence σ, can be fired in $Collapse(\mathcal{E}_1)$ as well, and leads from from the s_1^1 to s^1. By definition of the atomic parallel composition, the state (s^1, s_2^2) is reachable in $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$. As $s^2 = s_2^2$, we obtain again that $s = (s^1, s^2)$ is reachable in $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$.

($\Leftarrow$) If s is a stable state of $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$, then either s is an initial state of $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$, and we have seen that the initial states of $\mathcal{E}_1|\mathcal{E}_2$ and of $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$ are the same; or s is obtained by firing a sequence of transitions that constitutes a syntactical macro-step σ of $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$, starting from a stable state s_1 that is reachable in $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$ (and also in $\mathcal{E}_1|\mathcal{E}_2$, by the induction hypothesis). Then, either $\sigma = Collapse(\sigma')$, or $\sigma = \sigma'$, for σ' a syntactical macro-step of $\mathcal{E}_1$ or of $\mathcal{E}_2$. Then, we prove (just as in the proof of the ($\Rightarrow$) implication) that the effects of executing the sequence σ in $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$, and σ' in $\mathcal{E}_1|\mathcal{E}_2$, are the same. Now, σ leads from s_1 to s in $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2)$, thus, σ' leads from s_1 to s in $\mathcal{E}_1|\mathcal{E}_2$. By the induction hypothesis, s_1 it reachable in $\mathcal{E}_1|\mathcal{E}_2$, and we obtain that s is reachable in $\mathcal{E}_1|\mathcal{E}_2$. Thus, (†) is proved, and the proposition follows.