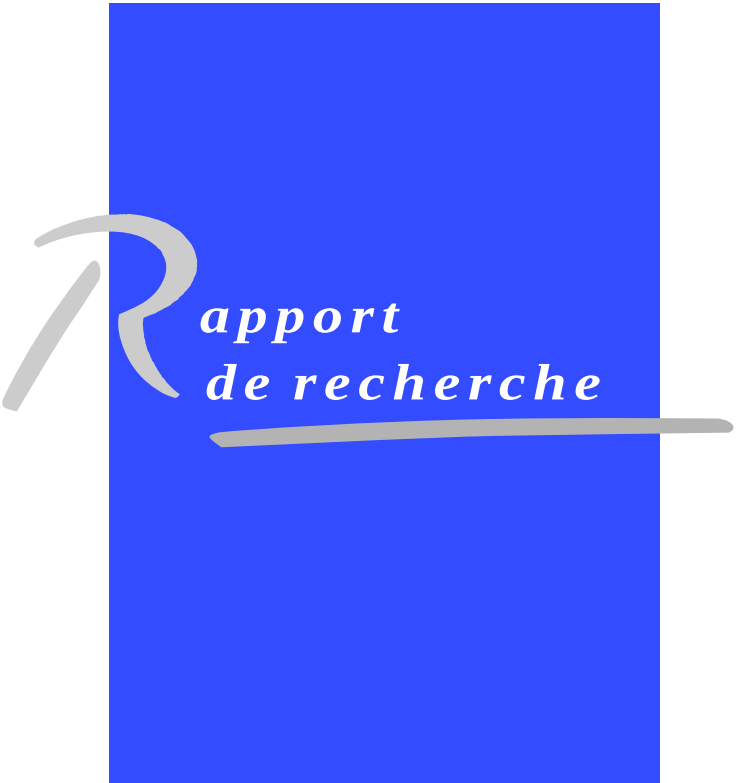
Verifying an ATM Protocol Using a Combination of Formal Techniques

Vlad Rusu

N°5089

2003

THÈME 1



ISSN 0249-6399 ISRN INRIA/RR--5089--FR+ENG

Projet Vertecs

Rapport de recherche n ° 5089 — 2003 — 39 pages

Abstract: 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: (1) a state-space exploration for initial debugging; (2) a partial-order abstraction that preserves the properties of interest; and (3) a compositional verification of the properties at the abstract level using the PVS theorem prover. Steps (2) and (3) guarantee that the properties still hold on the whole (composed, concrete) system as well. The value of the approach lies in adapting and integrating several formal techniques for verifying a real case study.

Key-words: partial-order abstraction, compositional deductive verification, PVS, SSCOP protocol.

(Résumé : tsvp)

A preliminary version of this paper has appeared in *Formal Methods Europe (FME'03)*, LNCS 2805, pages 223-243 [32].

The URL `http://www.irisa.fr/vertecs/Equipe/Rusu/sscop` contains PVS specifications and proofs for the case study.

* rusu@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

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 [4] 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 SSCOP protocol. The case study is the SSCOP (Service Specific Connection Oriented Protocol), an ATM protocol defined by the International Telecommunications Union. Among other uses, the protocol has been proposed as an alternative to TCP in datagram satellite networks [21].

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 [22], 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 [22]. It uses about twenty state variables of infinite types ranging from integers to complex types such as unbounded lists of records, whose fields may contain potentially unbounded lists.

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.

property.

Hence, we assume that some well-identified behaviours of the layers surrounding the protocol, which make the protocol violate its main property, do not occur. We prove that these necessary conditions are also sufficient for the protocol to satisfy its main property.

Partial-order abstraction. This abstraction consists in considering some sequences of actions (reading from a channel, then performing some internal actions, and finally writing to a channel) as *atomic*. This is a well-known technique used in finite-state model checking [1, 8, 25]. It is here adapted to the deductive verification of invariance properties on a class of infinite-state systems. The abstraction guarantees that the properties, once verified at the abstract level, will hold at the concrete level as well.

Compositional, deductive verification. For verifying invariants we use PVS [31], 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 to settle the initial goal.

This systematic *invariant strengthening* is performed in a (proved correct) *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.

Overall, we obtain a verification methodology for communication protocols, which is general enough to be applied with success to medium-sized system like the SSCOP protocol. The current case study took three months for a moderately experienced PVS user to complete.

The rest of the paper is organized as follows. In Section 2 a model of extended automata is defined, together with a method implemented in PVS for the verification of their invariance

2 Models

In this section the model of extended automata is presented (Section 2.1), together with a method implemented in PVS for proving invariance properties of this model (Section 2.2). The *collapsing* of an extended automaton is defined in Section 2.3. This operation produces an smaller, equivalent automaton (from a verification point of view) that is easier to verify.

2.1 Extended Automata

Definition 1 (Syntax of extended automata) *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, *which includes the initial locations* L_i, *and a set of* unstable *locations* $L_u = L \setminus L_s$,
- *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, which is a predicate on the variables* V,
  - *a vector of assignments* $(x := e_x)_{x \in V}$ *has exactly one assignment* $x := e_x$ *for each variable* $x \in V$. *Here,* e_x *is an expression of same type as* x,
  - *a destination location* l'.

whose result becomes observable for the environment only when a stable location is reached.

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

Notations and conventions. For a variable f of an array type, the expression $f[e_1] := e_2$ is an abbreviation for $f := \lambda k.\mathit{if}\ k = e_1\ \mathit{then}\ e_2\ \mathit{else}\ f(k)$. 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. For convenience, assignments of the form $x := x$ are not represented graphically.

Example 1 *The automaton $\mathcal{S}$ represented in Figure 1 (left) 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 $\mathfrak{L}$. 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 are used here to model potentially unbounded* FIFO *queues:*

- *$\mathfrak{R}$ models an* output *channel for the process $\mathcal{S}$, which* writes *to $\mathfrak{R}$ the value of variable x at position $\mathfrak{r}_w$.*
- *$\mathfrak{L}$ models an* input *channel for the process $\mathcal{S}$, which* reads *from $\mathfrak{L}$ 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 (right). This automaton has three locations, l'_0, l'_1, l'_2; the initial location l'_0 is the only stable location. It 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 $\mathfrak{L}$ 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 $\mathfrak{L}$ 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 $\mathfrak{R}$ (*Rightwards*) channel. The receiver $\mathcal{R}$ reads this sequence at the other*

of transitions $l_0 - l_1 - l_2 - l_0$ and $l_0 - l_1 - l_0$ are syntactical macro-steps.

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 $v(x)$, and by evaluating the result. In particular, if the type T is Boolean and $e(v)$ evaluates to *true* then v *satisfies the predicate* e.

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

Definition 4 (States, transition relations, 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 $\rho = \bigcup_{\tau \in \mathcal{T}} \rho_\tau$.*
- *A* run fragment *of the automaton is a sequence of states $r: \quad s_1, s_2, \ldots, s_n$ $(n \in \mathbb{N})$ 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 state predicate P is an* invariant *of the automaton $\mathcal{E}$, denoted $\mathcal{E} \models \Box P$, if P holds in every reachable state of $\mathcal{E}$.*

(formally, the implication $(P \wedge Q) \Rightarrow pre_t(P)$ is valid).

It is not hard to show that, if Q is an invariant, and if all transitions of the system preserve P using the auxiliary predicate 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. 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, or until evidence is obtained that P is not an invariant.

Example 2 *Consider the automaton $\mathcal{S}$ depicted in Figure 1. We want to prove that the predicate $\mathfrak{r}_r \leq \mathfrak{r}_w$ is an invariant of $\mathcal{S}$. For this, the automaton and the property are first translated into the language of* PVS. *Then, our* PVS *strategy for proving inductive invariants is applied, and here the strategy succeeds. This is because the predicate is inductive, i.e., it is true initially (when $\mathfrak{r}_r = \mathfrak{r}_w = 0$) and is preserved by all transitions of $\mathcal{S}$: the transition on the left-hand side increases $\mathfrak{r}_w$ by one, while that on the right-hand side does not modify $\mathfrak{r}_r, \mathfrak{r}_w$.*

Consider now the predicate $x \geq y$. This predicate is initially true, and is preserved by the left-hand side transition of $\mathcal{S}$ that increases x, but it is not *preserved by the right-hand side transition, which assigns to y the value $\mathfrak{L}[\mathfrak{l}_r]$, of which nothing is known at this point. Hence, the* PVS *strategy for proving inductive invariants fails, but it suggests to prove the auxiliary predicate $\forall i \in [\mathfrak{l}_r, \mathfrak{l}_w - 1].x \geq \mathfrak{L}[i]$, i.e., that x is greater or equal than every element in the channel $\mathfrak{L}$. Now, this predicate is true initially (when $\mathfrak{l}_r = \mathfrak{l}_w = 0$) and it is preserved by both transitions of $\mathcal{S}$: the left-hand side transition increases x, while the right-hand side transition removes an element from channel $\mathfrak{L}$, leaving fewer elements that x needs to be compared with. Hence, $\forall i \in [\mathfrak{l}_r, \mathfrak{l}_w - 1].x \geq \mathfrak{L}[i]$ and $x \geq y$ are invariants of $\mathcal{S}$*

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* $\sigma : \quad l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ *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* $\tau = \mathit{Collapse}(\sigma)$ *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 $(x := e_x)_{x \in V}$, 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 $\mathit{Collapse}(\sigma)$ is equivalent to that of σ. In Proposition 1, $\bullet$ denotes the composition of binary relations, defined as follows:

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

Proposition 1 *Let* $\sigma : \quad l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ *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* $\tau = \mathit{Collapse}(\sigma)$*. Then* (†): $\quad \rho = \rho_n \bullet \rho_{n-1} \bullet \cdots \bullet \rho_1$*.*

Proof. Cf. the Appendix, Page 33.

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

Proof. Cf. the Appendix, Page 34.

The collapsing operation applied to 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 extended automaton) *For $\mathcal{E}$ an extended automaton, the extended automaton Collapse($\mathcal{E}$) is the smallest structure built as follows:*

1. *the variables and initial condition of Collapse($\mathcal{E}$) are those of $\mathcal{E}$*
2. *for each maximal syntactical macro step σ : $l_1 \xrightarrow{\tau_1} l_2 \cdots l_n \xrightarrow{\tau_n} l_{n+1}$ of $\mathcal{E}$, Collapse($\mathcal{E}$) contains the transition Collapse(σ) (cf. Definition 5) and the locations l_1 and l_{n+1};*
3. *for every non-maximal syntactical macro step σ' : $l'_1 \xrightarrow{\tau'_1} l'_2 \cdots l'_m \xrightarrow{\tau'_m} l_{m+1}$, Collapse($\mathcal{E}$) contains all the transitions $\tau'_1, \ldots, \tau'_m$ as well as the all 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 2 of Definition 8 involves a finite set of operations; and Step 3 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 Fig. 1, the automaton *Collapse*($\mathcal{R}$) is depicted in the right-hand side of Fig. 2. For convenience, the automaton $\mathcal{S}$ is also depicted.

The main property of the *Collapse* operation is that it preserves reachability of stable states:

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

Proof. Cf. the Appendix, Page 34.

Here, $pc = l$ is the predicate that characterises the fact that control is at the location l. Then, proving the invariance of a stable predicate on an extended is equivalent (but, typically, easier, cf. Example 3 below) to 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 $\mathcal{E} \models \Box P$.*

Proof. Cf. the Appendix, Page 35.

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. Our collapsing procedure reduces the number of locations and, consequently, the number of proof obligations.

Example 3 *Consider the extended automaton $\mathcal{R}$ depicted in the right-hand side of Figure 1. We want to prove that $\varphi : \quad pc = l'_0 \Rightarrow \forall k \in [\mathfrak{r}_r, \mathfrak{r}_w - 1].z \geq \mathfrak{R}[k] - 1$ is an invariant of $\mathcal{R}$. This says that z is no less than $\mathfrak{R}[k] - 1$, for all integers $\mathfrak{R}[k]$ currently in the $\mathfrak{R}$ channel.*

By Proposition 4, establishing the invariance of φ on $\mathcal{R}$ is equivalent to establishing it on the collapsed automaton $\mathcal{R}' = Collapse(\mathcal{R})$, depicted in the right-hand side of Figure 2.

Now, φ is inductive on $\mathcal{R}'$; this is because φ holds initially (when the $\mathfrak{R}$ channel is empty), and it is preserved by both transitions of $\mathcal{R}'$, which remove *elements from the $\mathfrak{R}$ channel, leaving fewer elements $\mathfrak{R}[k]$ for which $z \geq \mathfrak{R}[k] - 1$ needs to be established (and moreover, the right-hand side transition increases z). Inductive invariants are quite easy to prove, e.g., our* PVS *strategy discussed in Section 2.2 establishes them automatically.*

On the other hand, φ is not inductive on $\mathcal{R}$, as the transition from l'_1 to l'_0 that decreases z does not preserve φ. In order to establish $\mathcal{R} \models \Box\varphi$, two auxiliary invariants φ_1 and φ_2 (not shown here) that describe the relations between z and the elements of $\mathfrak{R}$ in the (unstable) locations l'_1 and l'_2 are required, and then, $\varphi \wedge \varphi_1 \wedge \varphi_2$ must be proved inductive.

Hence, proving $Collapse(\mathcal{R}) \models \Box\varphi$ is typically easier than proving $\mathcal{R} \models \Box\varphi$, and the benefits of using the *Collapse* operation increase with the number of sequences that are collapsed.

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 (*semantical macro-steps*) to be executed atomically.

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 the composed system $\mathcal{E}_1||\mathcal{E}_2$ 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. $\mathcal{E}_2$).

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 ${s_2'}^{\downarrow V_2 \setminus V_1} = s_2^{\downarrow V_2 \setminus V_1}$ and $\varrho_1(s_1, s_1')$ holds.

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,

pendence 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 t a transition and s, s' states of an extended automaton, we write $s \xrightarrow{t} s'$ for $(s, s') \in \rho_t$.

Definition 10 *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be extended automata, t_1 a transition of $\mathcal{E}_1$, t_2 a transition of $\mathcal{E}_2$, and s a state of $\mathcal{E}_1||\mathcal{E}_2$. We say that* t_2 is independent of t_1 in state s *if for all states s', s'': $s \xrightarrow{t_1} s' \xrightarrow{t_2} s''$ implies that there exists a state $\tilde{s}$ such that $s \xrightarrow{t_2} \tilde{s} \xrightarrow{t_1} s''$. If t_2 is independent from t_1 and t_1 is independent from t_2 in state s, we say that* t_1 and t_2 are mutually independent in state s.

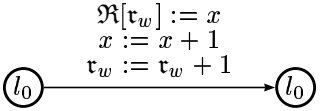
If t_2 is independent of t_1 in s, the execution of the sequence $t_1 t_2$ in s can be replaced by $t_2 t_1$, which produces the same global effect. This property is used in state-of-the-art model checkers [20] to fight the state-space explosion problem. In our theorem-proving framework, independence of transitions is employed for reducing the number of auxiliary proof obligations required for proving a given property.

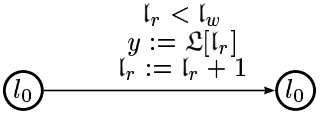
Proposition 5 below gives sufficient conditions for independence in reachable states of $\mathcal{E}_1||\mathcal{E}_2$. We use the following notations: let t be a transition of an extended automaton $\mathcal{E}$, and A denote its vector of assignments $(x := e_x)_{x \in V}$; we denote by $rhs(A)$ the vector of expressions $(e_x)_{x \in V}$. For t_1, t_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 $x \in V$.

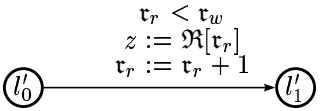
Finally, we define the dual *pre* of the weakest pre-condition $\widetilde{pre}$ (cf. Equation (1)) by

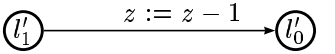
$$pre_t(P) : \exists s'.\rho_t(s, s') \wedge P(s') \tag{2}$$

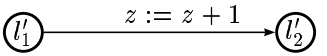
as follows:

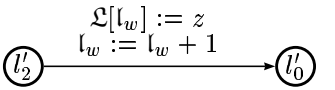
1. t_1 *denotes:* 

and t_2 *denotes:* 

2. t_3 *denotes:* 

and t_4 *denotes:* 

3. t_5 *denotes:* 

and t_6 *denotes:* 

We show using Proposition 5 that t_1 is independent of t_3 *in all reachable states of* $\mathcal{S}||\mathcal{R}$.

Hypothesis (2.) is easy to check, as all the variables that are syntactically modified by t_1 *(i.e.,* $\mathfrak{R}$*,* x*, and* $\mathfrak{r}_w$*) are syntactically not modified by* t_3*, and reciprocally.*

We now check Hypothesis (1.). The guard of t_3 *is* $G_3 : \quad \mathfrak{r}_r < \mathfrak{r}_w$*. Then, we have* $pre_{t_1}(G_3) : \quad \mathfrak{r}_r \leq \mathfrak{r}_w$*. It has been shown in Example 2 that both transitions (*t_1 *and* t_2*) of* $\mathcal{S}$ *preserve the predicate* $\mathfrak{r}_r \leq \mathfrak{r}_w$*, and it is not hard to check that the predicate is preserved by all the transitions of* $\mathcal{R}$ *(*$t_3, \ldots, t_6$*) as well. Hence,* $pre_{t_1}(G_3)$ *is also an invariant of* $\mathcal{S}||\mathcal{R}$*, which implies Hypothesis (1.). Then, by Proposition 5,* t_1 *is independent of* t_3 *in all reachable states of* $\mathcal{S}||\mathcal{R}$*. It can be shown in a similar manner that* t_6 is independent of t_2.

On the other hand, t_1 is mutually independent with t_4, t_5, t_6*; and* t_2 is mutually independent with t_3, t_4, t_5*. These mutual independences can be proved directly using a syntactical argument instead of Proposition 5: the transitions operate on disjoint sets of variables.*

Finally, t_3 *is not independent of* t_1 *in the initial states (where* $\mathfrak{r}_r = \mathfrak{r}_w = 0$*): the sequence* $t_1 t_3$ *can be executed, but the sequence* $t_3 t_1$ *cannot, as the guard* $G_3 : \quad \mathfrak{r}_r < \mathfrak{r}_w$ *is* false.

decomposition $\sigma = \sigma' \cdot \sigma''$ of syntactical macro-steps (cf. Definition 11) can be employed:

- 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 *output* to the other channel.

Indeed, Definition 11 requires that every transition (except may 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 satisfying this condition.

Example 5 *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 $t_1 \ldots t_6$ as in Example 4, $\mathcal{S}$ has two syntactical macro-steps, denoted by $\sigma_1^1 : \quad l_0 \stackrel{t_1}{\rightarrow} l_0$ and $\sigma_2^1 : \quad l_0 \stackrel{t_2}{\rightarrow} l_0$; and $\mathcal{R}$ also has two syntactical macro-steps, $\sigma_1^2 : \quad l'_0 \stackrel{t_3}{\rightarrow} l'_1 \stackrel{t_4}{\rightarrow} l'_0$ and $\sigma_2^2 : \quad l'_0 \stackrel{t_3}{\rightarrow} l'_1 \stackrel{t_5}{\rightarrow} l'_2 \stackrel{t_6}{\rightarrow} l'_0$. Consider the following decompositions of these sequences, where ε denotes the empty sequence:*

- $\sigma_1^1 = {\sigma'}_1^1 \cdot {\sigma''}_1^1$ *with* ${\sigma'}_1^1 = \varepsilon$ *and* ${\sigma''}_1^1 = l_0 \stackrel{t_1}{\rightarrow} l_0$;
- $\sigma_1^2 = {\sigma'}_1^2 \cdot {\sigma''}_1^2$ *with* ${\sigma'}_1^2 = l_0 \stackrel{t_2}{\rightarrow} l_0$ *and* ${\sigma''}_1^2 = \varepsilon$;
- $\sigma_1^2 = {\sigma'}_1^2 \cdot {\sigma''}_1^2$ *with* ${\sigma'}_1^2 = l'_0 \stackrel{t_3}{\rightarrow} l'_1$ *and* ${\sigma''}_1^2 = l'_1 \stackrel{t_4}{\rightarrow} l'_0$;
- $\sigma_2^2 = {\sigma'}_2^2 \cdot {\sigma''}_2^2$ *with* ${\sigma'}_2^2 = l'_0 \stackrel{t_3}{\rightarrow} l'_1$ *and* ${\sigma''}_2^2 = l'_1 \stackrel{t_5}{\rightarrow} l'_2 \stackrel{t_6}{\rightarrow} l'_0$.

Using the independence relations between transitions $t_1 \ldots t_6$ that have been established in Example 4, it can be easily shown that these decompositions satisfy Definition 11, i.e., the automata $\mathcal{S}$ and $\mathcal{R}$ from Figure 1 are mutually independent.

Definition 12 (Stable states, projection) *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 $\mathcal{E}_2$.*

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 as well.

In this section we provide a compositional rule for verifying invariants at the abstract level, and how this rule can be combined with the collapsing operation defined in Section 2.3 for reducing the number of auxiliary proof obligations required by a given verification task.

The compositional rule (cf. Proposition 8 below) says essentially that the $|$-composition of two automata satisfies a given invariant whenever the two automata, suitably modified, satisfy the invariant as well. 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 *$\mathcal{E}$ $\mathbf{init}\langle P, L\rangle$ 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, $\mathcal{S}$ $\mathbf{init}\langle x \geq y, \{l_0\}\rangle$ 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 for the $|$ parallel composition) *Let $\mathcal{E}_1, \mathcal{E}_2$ be extended automata 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 as well.

Proof. Cf. the Appendix, Page 37.

Example 6 *Consider the parallel composition $\mathcal{S}|\mathcal{R}'$ of the extended automata depicted in Figure 2. We want to prove that $P: \quad x \geq y$ is an invariant of this system. By Proposition 8 it is enough to prove that the predicate holds initially in $\mathcal{S}$ and $\mathcal{R}$ (which is trivially true), and to prove (1) $\mathcal{S}$ $\mathbf{init}\langle x \geq y, l_0, \rangle \models x \geq y$, and (2) $\mathcal{R}'$ $\mathbf{init}\langle x \geq y, l'_0\rangle \models x \geq y$. Now, (2) is trivial, because $\mathcal{R}'$ does not modify x, y. For the proof of (1), we have already seen in Example 2 that P is not inductive, and that the failure of* PVS *suggests to prove the auxiliary invariant $Q : \forall i \in [\mathfrak{l}_r, \mathfrak{l}_w - 1].x \geq \mathfrak{L}[i]$. Hence, we obtain two new proof obligations: (1') $\mathcal{S}$ $\mathbf{init}\langle P \wedge Q, l_0\rangle \models P \wedge Q$, which is trivial, because Q was introduced precisely for this; and (2') $\mathcal{R}'$ $\mathbf{init}\langle P \wedge Q, l'_0\rangle \models P \wedge Q$. Next, applying the* PVS *proof strategy for (2') results in failure. Iterating this process a few times suggests to prove that the auxiliary predicate φ:*

$$\forall (i,j,k,l) \in [\mathfrak{l}_r, \mathfrak{l}_w - 1] \times [\mathfrak{l}_r, i] \times [\mathfrak{r}_r, \mathfrak{r}_w - 1] \times [\mathfrak{r}_r, k].\, x \geq \mathfrak{R}[i]+1 \geq \mathfrak{R}[j]+1 \geq \mathfrak{L}[k] \geq \mathfrak{L}[l] \geq y$$

(which implies the original goal $x \geq y$) satisfies $\mathcal{S}$ $\mathbf{init}$ $\langle \varphi, l_0\rangle \models \Box\varphi$ and $\mathcal{R}'$ $\mathbf{init}$ $\langle \varphi, l_0\rangle \models \Box\varphi$. These proof obligations are automatically established by our PVS *strategy, because φ is preserved by all transitions of $\mathcal{S}$ and $\mathcal{R}'$; moreover, φ also holds initially in $\mathcal{S}$, $\mathcal{R}'$.*

Hence, by Proposition 8, $\mathcal{S}|\mathcal{R}' \models \Box\varphi$ holds, which implies our initial goal $\mathcal{S}|\mathcal{R}' \models \Box(x \geq y)$.

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 (†) $pc_1 = l \Rightarrow Q'$, where $l \in L^1_u$ 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, a predicate Q of the form (†) trivially holds in the initial states of $\mathcal{E}_1$ $\mathbf{init}$ $\langle P, L^1_s\rangle$. Thus, proving $\mathcal{E}_1$ $\mathbf{init}$ $\langle P \wedge Q, L^1_s\rangle \models \Box(P \wedge Q)$ reduces to proving $\mathcal{E}_1$ $\mathbf{init}$ $\langle P, L^1_s\rangle \models \Box(P \wedge Q)$. Hence, if one also proves $\mathcal{E}_2$ $\mathbf{init}$ $\langle P, L^2_s\rangle \models \Box P$ (and P holds initially in $\mathcal{E}_2$) then the compositional rule (Proposition 8) can be used to deduce $\mathcal{E}_1|\mathcal{E}_2 \models \Box P$; thus, a proof that Q is preserved by $\mathcal{E}_2$ has been saved.

for stable predicates, the compositional rule (Prop. 8) can be equivalently applied to the *collapsed* 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 $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2) \models \Box P$.*

Proof. Cf. Appendix, Page 38.

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 $Collapse(\mathcal{E}_1)\mathbf{init}\langle P, L_s^1\rangle \models \Box P$ and $Collapse(\mathcal{E}_2)\mathbf{init}\langle P, L_s^2\rangle \models \Box P$ hold. Hence, $\mathcal{E}_1$ and $\mathcal{E}_2$ can be collapsed prior to using the compositional rule. As shown in, e.g,. Example 3 at the end of Section 2.3, this may significantly reduce the number of proof obligations required to complete a proof.

Summary: General Verification Methodology

We now combine the results obtained in Sections 2, 3, and 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 5 and/or simple syntactical arguments (when the transitions access disjoint sets of variables).

Then, the automata are collapsed, and the compositional rule (Proposition 8) is used to establish $Collapse(\mathcal{E}_1)|Collapse(\mathcal{E}_2) \models \Box P$, using PVS and the invariant-strengthening techniques described in Section 2.2. If the verification succeeds, by Propositions 4 and 9, we obtain $\mathcal{E}_1|\mathcal{E}_2 \models \Box P$.

Finally, Proposition 7 implies that $\mathcal{E}_1||\mathcal{E}_2 \models \Box P$ holds as well.

The SSCOP protocol consists of a sender and a receiver that exchange Protocol Data Units (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., [6] for the verification of this process by model checking). Then, 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, which is increased by one with every new DATA sent;

- 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.

the effect that the protocol violates its main property. For example, the receiver's client may require disconnection, and when this happens, a state is entered where the incomplete subsequence of data received so far may be retrieved by the receiver's client:

Assumption 1: the upper layer does not disrupt an ongoing data transfer on purpose.

Assumptions have to be made on the lower layer as well: the communication channels may only lose, but not create, duplicate, or reorder PDUs. This is because creation, duplication, or reordering of PDUs may also prevent the protocol from satisfying its main property.

For example, if an 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.

Assumption 2: the lower layer may only lose *but not 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 (the subsequence of data received so far is typically incomplete):

Assumption 3: the connection is not lost because of too many losses by the lower layer.

Formally modeling the protocol. The protocol is modeled as SSCOP:= *Sender*||*Receiver*, where || is the *interleaved* parallel composition (cf. Section 2). This is consistent with the semantics of the protocol originally expressed in SDL [22], 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, 3, and 4).

A brief description of the sender and receiver processes follows. (The reader may consult [6, 22] for more details.) Both processes can be decomposed into several components, which correspond to SDL "transitions" in the specification [22]. An SDL transition is similar to

Sending DATA and POLL.

This component is in charge of sending new DATA PDUs as well as old ones (retransmissions) and of interrogating the receiver by means of POLL PDUs. New DATA PDUs are taken from the *Request* sequence (cf. Figure 3), whereas old ones are taken from the retransmission queue *ReXmitQueue*. Each new DATA is given a unique index number and is sent on the SR channel, and its data and index are memorized in the sender's window *XmitBuffer*, together with the current poll sequence number $VTPS$. The latter is a variable whose initial value is zero and which is incremented each time a new POLL PDU is sent by the current component. This happens every $MaxPD \geq 1$ transmissions of new DATA, and after every retransmission of an old DATA.

A POLL PDU carries, among other values, the current values of $VTPS$ and of VTS. The latter denotes the highest index of an DATA PDU that the sender has ever emitted.

This component has two stable locations: the initial location, and a final location which is entered when all DATA PDUs have been acknowledged by USTAT or STAT PDUs. In the final location, the sender is idle.

Receiving USTAT.

USTAT PDUs are sent by the receiver of the SSCOP upon reception of a DATA PDU with an unexpectedly high index number. Technically, this happens when the index of the received DATA PDU is higher than the highest index the receiver knows about. For the sake of precision, let us name VRH the latter index, and VRR the highest index such that the all messages up to VRR were correctly received in sequence.

Then, a USTAT consists of a copy of the receiver's state variables, including: VRH; the index of the message just received; and VRR. The interval defined by the first two values constitutes a set of DATA PDU the receiver assumes to be *lost*; on the other hand, all DATA with index below VRR have to be acknowledged.

This is the third and last component of the sender and is somewhat similar to receiving USTAT. A STAT PDU is generated by the receiver of the SSCOP upon reception of a POLL PDU. It consists of copies of the state variables of the receiver, mainly, an abstract view of the receiver's window.

Upon reception of STAT PDU, the sender checks its validity just like for a USTAT, with the same outcome if it is found invalid, i.e., exit from the data transfer mode, modeled by a location called *OutOfDtr*. Otherwise, the copy of the receiver's window is scanned, and the DATA PDUs that it describes as missing are added to the retransmission queue of the sender. To avoid useless retransmission, a DATA PDU will not be added to the retransmission queue if it is already there. The same happens if the *VTPS* value memorized together with the index of the DATA in the sender's window *XmitBuffer* (cf. "*Sending* DATA *and* POLL" paragraph) is above a certain threshold of acceptable values. This rather intricate mechanism is crucial for ensuring the property that no retransmissions occur unless really necessary, a property that the protocol consistently enforces for efficiency considerations. This component has one stable location (the initial location)

5.2 The Receiver

The receiver is modeled as *Receiver* := (DATA *receiver*|POLL*receiver*) and its behaviour is explained below.

Receiving DATA.

Upon reception of an DATA PDU, the receiver checks if the DATA was already received, i.e., if it is already memorized in the receiver's window *RecvBuffer*. If this is the case, the data transfer mode is terminated, which we model by the *OutOfDtr* location that we shall prove unreachable. Otherwise, if the index of the DATA PDU is between the next expected index and the highest index ever received (called respectively *VRR* and *VRH*, cf. "receiving USTAT" paragraph), the DATA is stored in the receiver's window, and the lower bound of the window

This component and the verification of an invariant connected to it are described in more detail in Section 5.4.

5.3 Verification: the Main Invariants

The system to be verified is the interleaved parallel composition *Sender*||*Receiver*, and the main property is expressed using a *stable* predicate (cf. Definition 9), saying that whenever both sender and receiver are in the *Idle* 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 a few simple invariants, in a manner completely similar to what was described in Section 3.2 (cf. Examples 4 and 5) for the running example of the paper. 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. This leads to significant savings in terms of the number of auxiliary invariants required to complete a proof.

Moreover, we proceed incrementally, starting with three components only: the DATA sender, the DATA receiver, and the USTAT receiver. After having established a number of invariants of their composition, the two remaining components (POLL receiver and STAT receiver) are added, and we show that they preserve all the invariants previously established.

Verifying the composition (DATA & POLL *sender* | USTAT *receiver*) | (DATA *receiver*)**.** Rather than starting with the main property of the protocol, we start with the predicates stating that the *OutOfDtr* locations are unreachable. The advantage is that these predicates directly involve the internal state variables of the sender and receiver, while the higher-level main property does not. This, in turn, suggests to prove facts such as:

mutually redundant mechanisms for requesting retransmissions: the USTAT and STAT PDUs, have to interact while preserving the property that DATA PDUs are not retransmitted unless they were really lost, a property that the protocol consistently enforces.

5.4 Verifying One Representative Invariant

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

† *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 (†). The property is about STAT PDUs, which are generated by the POLL receiver component in response to a POLL PDU. The component (the smallest of all five in our system) is depicted, 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 denote the values of the sender's VTS and $VTPS$ variables at the time the POLL was emitted. Remember (cf. "sending DATA and POLL" paragraph) that VTS is the highest index of an DATA PDU that the sender ever 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 (not shown here) allows to prove that this location is unreachable.

Figure 4: The POLL receiver component.

Otherwise, *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:

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

Hence, the actual encoding of statement (†) 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 (‡) above. Our dedicated PVS strategy is applied again, which suggests to prove the two predicates depicted in Fig. 7 are invariants. The latter are inductive and are automatically proved by our strategy, which implies that auxiliary predicate depicted in Fig. 6 is an invariant, which in turn implies that the predicate depicted in Fig. 5 is an invariant as well.

Discussion. Note that the verification has only generated auxiliary invariants (cf. Figures 6, 7) that are *unstable* according to Definition 9 and our choice of stable locations for

```
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 used for proving `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.

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 [22] 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 three 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 took several months as well. 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.

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. On the other

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 to incorporate the partial-order reductions in the model-checking algorithms.

Historically, symmetrical independence relations (i.e., if t_1 is independent of t_2, then t_2 is independent of t_1 as well) have been the first employed. The article [25] studies their fundamental properties. Applications to model checking with on-the-fly detection of independent transitions are presented in the monograph [8]. The SPIN model checker [20] uses partial orders intensively, sometimes with dramatic effects on the reduction of the state-space explored. Asymmetric independence relations (t_1 may be independent of t_2, but the converse may not necessarily be the case) have been introduced more recently [1].

The independence relation that we employ in this paper (Definition 10) paper 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 relation was well suited to the present theorem-proving framework, because establishing it generated little verification overhead (a few simple invariants).

The article [15] 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. By 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 be more efficient and to involve the same basic routine with the prover. Compositionality is not considered in [15].

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 [24].

The interested reader may consult the reference [12] for all things compositional and non-compositional. The second volume, in preparation, promises a deep embedding of assume-guarantee reasoning in PVS.

Verification of communication protocols

Relevant works in the deductive verification of communication protocols include [3, 9, 10, 18, 19, 28]. 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 [3] was performed in COQ [11] and took three months. The deductive verification of the same protocol with the more automated PVS system took one month [18]. Using the most recent features of abstraction and model checking in PVS [33] the protocol was automatically verified in a matter of seconds.

The automatic verification of protocols specified in SDL has also received some attention recently [5, 17, 23, 34]. 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. Compositionality and

[2] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, S. Tasiran. Mocha: modularity in model checking. *Computer-Aided Verification*, LNCS 1427, 1998.

[3] M. Bickford, C. Kreitz, R. van Renesse, X. Liu. Proving hybrid protocols correct. *Theorem Proving in Higher Order Logics*, LNCS 2152.

[4] J. Burch, E. Clarke, K. McMillan, D. Dill, J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[5] D. Bosnacki, D. Dams, L. Holenderski, N. Sidorova. Model checking SDL with Spin. *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1785, 2000.

[6] M. Bozga, J.-C. Fernandez, L. Ghirvu, C. Jard, T. Jéron, A. Kerbrat, P. Morel, L. Mounier. Verification and test generation for the SSCOP protocol. *Science of Computer Programming* 36(1):27–52, 2000.

[7] S. Bensalem, Y. Lakhnech, S. Owre. Constructing abstractions of infinite state systems compositionally and automatically. *Computer-Aided Verification*, LNCS 1427, 1998.

[8] B. Boigelot. Partial-order methods for the verification of concurrent systems. LNCS 1032.

[9] R. Cardell-Oliver. On the use of the HOL system for protocol verification. *International Workshop on the HOL Theorem Proving System and its Applications*, IEEE Computer Society, 1992.

[10] D. Chkliaev, J. Hooman, E. de Vink. Verification and improvment of the sliding-window Protocol. *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2619, 2003.

Verification (CAV'99), LNCS 1633.

[16] M. Gordon, T. Melham. *Introduction to the HOL system.* Cambridge University press, 1994.

[17] S. Graf, G. Jia. Verification experiments on the MASCARA protocol. *SPIN workshop on software model checking*, 2001.

[18] K. Havelund, N. Shankar. Experiments in theorem proving and model checking for protocol verification. *Formal Methods Europe*, LNCS 1051, 1996.

[19] L. Helmink, M. Sellink, F. Vaandrager. Proof-checking a data link protocol, *Types for Proofs and Programs*, LNCS 806, 1993.

[20] G. Holzmann. The model checker Spin. IEEE Transactions on Software Engineering, 23(5): 279–295, 1997.

[21] T. Henderson, R. Katz. STP: a SSCOP-based transport protocol for Datagram Satellite Networks. *Intl. Workshop on Satellite-based Information Services*, 1997.

[22] International Telecommunication Union. ATM Adaptation Layer - Service Specific Connection Oriented Protocol. *Recommendation Q.2110*, 1994.

[23] N. Ioustinova, N. Sidorova, M. Steffen. Closing open SDL-systems for model checking with DTSpin. *Formal Methods Europe*. LNCS 2391, 2002.

[24] C.B. Jones. Tentative steps towards a development method for interfering programs. *ACM Transactions on Programming Languages and Systems,* 5(4):596-619, 1983.

[25] S. Katz, D. Peled. Defining conditional independence using collapses. *Theoretical Computer Science*, 101:227-359, 1992.

Engineering, 21(2):107-125, 1995.

[32] V. Rusu. Compositional verification of an ATM protocol. *Formal Methods Europe*, LNCS 2805, 2003.

[33] H. Saïdi, N. Shankar. Abstract and model check while you prove. *Computer-Aided Verification*, LNCS 1633, 1999.

[34] N. Sidorova, M. Steffen. Verifying large SDL specifications using model checking *SDL Forum*, LNCS 2078, 2001.

($\subseteq$) 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:

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

- $(s, s'') \in \rho'$, as the location l_1 of s is the origin of τ'; the valuation v of s satisfies G' (cf. (1.)); the location l_n of s'' is the destination of τ'; and the valuation of s'' is $A'(v)$,
- $(s'', s') \in \rho_n$, 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. (1.) 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 $G = G' \wedge G_n \circ A'$, v satisfies G,

(iv) $s' = \langle l_{n+1}, v'\rangle$ for the valuation $v' = A_n(v'') = (A_n \circ A')(v)$. As $A = A_n \circ A'$, $v' = A(v)$.

given graph), or σ contain a cycle that does include a *stable* location. Now, by Definition 3, 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 Collapse($\mathcal{E}$) its collapsed automaton. Then, a stable state is reachable in $\mathcal{E}$ if and only if it is also reachable in 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 *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 $r'' : \quad s_1, \cdots, s_n = s$ 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 $\sigma : l_1 \stackrel{\tau_1}{\rightarrow} \cdots \stackrel{\tau_{n-1}}{\rightarrow} l_n$ is a *syntactical* macro step, which is either:

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 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 $\mathcal{E}$ as well. By induction hypothesis, the state s_1 it reachable in $\mathcal{E}$, and we obtain that s is reachable in $\mathcal{E}$ as well. □

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 3 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, t_1 a transition of $\mathcal{E}_1$, and t_2 a transition of $\mathcal{E}_2$. Let G_1 denote the guard of t_1, l_1 denote the origin of t_1, and A_1, A_2 denote respectively the assignments of t_1, t_2. Then, t_2 is independent of t_1 in every reachable state of $\mathcal{E}_1 || \mathcal{E}_2$ if*

(1.) $\mathcal{E}_1 || \mathcal{E}_2 \models \Box[(pc_1 = l_1 \wedge G_1) \Rightarrow pre_{t_2}(G_1)]$ *and*

(2.) $\mathcal{E}_1 || \mathcal{E}_2 \models \Box[(pc_1 = l_1 \wedge G_1) \Rightarrow rhs(A_1 \circ A_2) = rhs(A_2 \circ A_1)]$

Proposition 6 *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be mutually independent extended automata, and r a run of $\mathcal{E}_1||\mathcal{E}_2$ that terminates in a stable state s. Then, there exists a run $\tilde{r}$ of $\mathcal{E}_1|\mathcal{E}_2$ that terminates in the state s as well.*

Proof. 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.

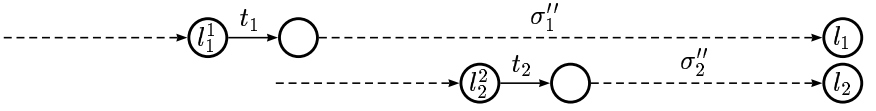
For the induction step, assume that the length of the run r is $n \geq 1$, and let s^1 be a predecessor of s on the run r such that (a) the projection s_1^1 of s^1 on $\mathcal{E}_1$ is stable, and (b) there is no other state on r between s^1 and s whose projection on $\mathcal{E}_1$ is stable.

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 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 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 t_1 of $\sigma_1 = t_1 \cdot \sigma_1''$ before $\mathcal{E}_2$ fires the first transition t_2 of $\sigma_2 = t_2 \cdot \sigma_2''$. This situation is depicted as follows:



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 as well.*

Proof. ($\Rightarrow$) Let $P: \quad pc_1 = l_1 \wedge pc_2 = l_2 \Rightarrow P'$. 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 6 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 $\mathcal{E}_1||\mathcal{E}_2$ $\Box$

Proposition 8 *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be extended automata 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 as well.

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 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. $\Box$

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 $r'' : \quad s_1, \cdots, s_n = s$ between s_1 and s, which is a semantical macro-step of $\mathcal{E}_1|\mathcal{E}_2$.

By 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,

1. 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.
2. if σ is *not* maximal, then by Definition 3, the sequence σ appears in $Collapse(\mathcal{E}_1)$ as well. 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

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399