# Atomicity Refinement and Trace Reduction Theorems

E. Pascal Gribomont
Institut Montefiore, Université de Liège, Sart-Tilman B 28, B-4000 Liège (Belgium)
gribomon@montefiore.ulg.ac.be

**Abstract.** Assertional methods tend to be useable for abstract, coarse-grained versions of concurrent algorithms, but quickly become intractable for more realistic, finer-grained implementations. Various trace-reduction methods have been proposed to transfer properties of coarse-grained versions to finer-grained versions. We show that a more direct approach, involving the explicit construction of an (inductive) invariant for the finer-grained version, is theoretically more powerful, and also more appropriate for computer-aided verification.

## 1 Introduction

Recents improvements in methods and tools for testing the validity of propositional and predicate logic formulas have revived the interest in assertional methods for concurrent system verification. Indeed, at least as far as safety properties are concerned, Hoare's logic and Dijkstra's predicate transformer calculus reduce the correctness problem for programs to the validity problem for logical formulas.

However, as soon as loops occur in programs, creativity is needed to discover appropriate invariants. This task is reasonably feasible for coarse-grained, abstract concurrent systems, but often becomes intractable for fine-grained, reasonably efficient implementations.

A standard technique is to deal first with a coarse-grained version of the system to be verified, and then to attempt (in a more or less formal way) to adapt the conclusion to a finer-grained implementation of the system. This is called *atomicity refinement*. In this paper, we compare two frequently used techniques for atomicity refinement, from both theoretical and practical point of view.

The problem solved by these techniques is as follows. Some concurrent system has been proved correct with respect to some safety property. Some statement is replaced by an equivalent sequence of more elementary statements. Due to possible interference between processes, this atomicity refinement is not always correct. How can such a refinement be validated (or disproved)? Let us consider a two-process system **S**, where the (cyclic) concurrent processes are

$$Loop(S_1; S_2) \quad \text{and} \quad Loop(T_1; T_2)$$

There is some initial condition $A$ and some safety property $J$, validated with some invariant $I$. Otherwise stated, there is an assertion $I$ such that $A \Rightarrow I$, $I \Rightarrow J$, and, for each state $\sigma$ satisfying $I$, if any of the transitions $S_1$, $S_2$, $T_1$ and $T_2$ can be executed from state $\sigma$, then the resulting state $\rho$ also satisfies $I$. As a consequence, any **S**-computation whose initial state satisfies $A$ reaches only states satisfying $J$.

Now we replace a transition, say $T_2$, by an equivalent sequence, say $T'; T''$ (transition $T_2$ can lead from state $\sigma_1$ to state $\sigma_2$ if and only if sequence $T'; T''$ can lead from $\sigma_1$ to $\sigma_2$). The question is, is the new system **S'** still correct w.r.t. the safety property $J$?

There is clearly no problem with *primary* **S'**-computations, such that any execution of $T'$ is *immediately* followed by an execution of $T''$, without interference from

$S_1$ or $S_2$. Let us call $B$ the assertion which holds in all states but those "between" some execution of $T'$ and the corresponding execution of $T''$. It is clear that $J$ still holds in *relevant states* (those satisfying $B$), that is, that $B \Rightarrow J$ remains true throughout the computation.

Now, let us consider the general case where some execution(s) of $S_1$ and $S_2$ take(s) place between an execution of $T'$ and an execution of $T''$, for instance

$$S_1; T_1; T'; S_2; T''; T_1; S_1; T'; S_2; S_1; T''; \ldots$$

It is not always the case that $B \Rightarrow J$ remains true throughout the computation.

The *trace reduction method* guarantees that $B \Rightarrow J$ remains a safety property, provided that $T'$ is a *right-mover*, i.e., the following holds: if $(T'; S_1)$ can lead from some state $\sigma$ to some state $\rho$, then $(S_1; T')$ can also lead from $\sigma$ to $\rho$, and the same with $S_1$ replaced by $S_2$. (Instead of requiring $T'$ to be a right-mover, we can require $T''$ to be a *left-mover*.) This method is of easy application and has led to successful non-trivial designs; it is especially useful to convert centralized concurrent systems into distributed ones. The drawback is that the method is not complete; some correct atomicity refinements cannot be validated that way.

The *invariant adaptation method* consists in finding some invariant $I'$ of $\mathbf{S}'$ which reduces to $I$ in every relevant state. This method is complete in the following sense: if $J$ is a safety property of $\mathbf{S}$ that remains true in all relevant states of $\mathbf{S}'$, then adequate invariants $I$ and $I'$ exist. The knowledge of $I$ is a big help for the construction of the adapted invariant $I'$, but this construction often turns to be a complicated task nevertheless.

A usual policy for validating atomicity refinements is therefore to try the trace reduction method first, and, only in case of failure, to try the invariant adaptation method. The purpose of this paper is to show that success cases for the reduction method always are elementary cases for the invariant adaptation method, whereas some elementary cases for the invariant adaptation method are still failure cases for the reduction method. As a result, it might be better to use only the invariant adaptation method, especially for computer-aided design/verification.

The paper goes on as follows. An abstract framework for atomicity refinement is introduced in Section 2, where the trace reduction method is presented as a special case of the invariant adaptation method. Both methods are compared in a more general way in Section 3, where success cases for the trace reduction method are proved to correspond to cases of easy invariant adaptation. Section 4 shows that a failure case for the reduction method can turn to be an easy case for the invariant adaptation method. Section 5 is a conclusion and mentions related works.

## 2 Theorems about atomicity refinement

We introduce an abstract framework for atomicity refinement and show that, from the theoretical point of view, the trace reduction method is a particular case of the invariant adaptation method. More specifically, we recall the main theorem about trace reduction and give a theorem connecting the invariant of a system before and after the atomicity refinement. The former appears as a mere corollary of the latter.

## 2.1 Relational notation

Let $\mathcal{R}$ and $\mathcal{S}$ be binary relations on a non-empty set $\Gamma$, and let $\gamma \in \Gamma$ and $A \subseteq \Gamma$. The following notation is used in the sequel.

$$\begin{array}{ll}
\mathbf{1}_\Gamma =_{def} \{(\gamma,\gamma) : \gamma \in \Gamma\}, & \text{(identical relation)}, \\
\mathcal{R};\mathcal{S} =_{def} \{(\gamma,\delta) : \exists \rho\, [(\gamma,\rho) \in \mathcal{R} \wedge (\rho,\delta) \in \mathcal{S}]\}, & \text{(sequential composition)}, \\
\mathcal{R}^0 =_{def} \mathbf{1}_\Gamma,\ \mathcal{R}^{n+1} =_{def} (\mathcal{R}^n;\mathcal{R}),\ \mathcal{R}^* =_{def} \bigcup_{n\geq 0} \mathcal{R}^n, & \text{(iteration, closure)}, \\
\gamma\mathcal{R} =_{def} \{\delta : (\gamma,\delta) \in \mathcal{R}\},\ A\mathcal{R} =_{def} \bigcup_{\rho\in A} \rho\mathcal{R}, & \text{(set of successors, postset)}.
\end{array}$$

*Comments.* A binary relation on $\Gamma$ is simply a subset of $\Gamma \times \Gamma$. The notation $\gamma\mathcal{R}\delta$ usually stands for $(\gamma,\delta) \in \mathcal{R}$. The sequential composition $\mathcal{R};\mathcal{S}$ is also noted $\mathcal{S} \circ \mathcal{R}$. Note that $A\mathcal{R}\mathcal{S} = A(\mathcal{R};\mathcal{S})$. An element $\gamma$ is an $\mathcal{R}$-*predecessor* of $\delta$ if $\delta$ is an $\mathcal{R}$-successor of $\gamma$, that is, if $(\gamma,\delta) \in \mathcal{R}$.

## 2.2 Abstract transition systems

An *abstract transition system* [18, 28] is a couple $\mathbf{Ats} = (\Gamma, \{\mathcal{R}_1,\ldots,\mathcal{R}_n\})$ where $\Gamma$ is a non-empty *state space* and where $\{\mathcal{R}_1,\ldots,\mathcal{R}_n\}$ is a finite non-empty set of *actions*, i.e., binary relations on $\Gamma$. A *state* is an element $\gamma \in \Gamma$. A *predicate* is a subset $A \subseteq \Gamma$.

*Comment.* Predicates are usually represented as assertions, so we will write $\gamma \models A$ ("$\gamma$ satisfies $A$", "$A$ is true at $\gamma$") instead of $\gamma \in A$. Similarly, we write $\neg A$, $A \wedge B$ and $A \vee B$ instead of $\Gamma \backslash A$, $A \cap B$ and $A \cup B$, respectively. An assertion $C$ is *valid* if the corresponding set is $\Gamma$; we write $\models C$ instead of $(\forall \gamma \in \Gamma)\,(\gamma \models C)$; the inclusion $A \subseteq B$ therefore becomes $\models (A \Rightarrow B)$. Last, the successor set $A\mathcal{R}$ and the reachability set $A\mathcal{R}^*$ are modelled by the assertions $sp[A;\mathcal{R}]$ ("strongest postcondition") and $sin[A;\mathcal{R}]$ ("strongest invariant") respectively.

An $(\mathbf{Ats}, A)$-*traced computation*, or simply a *traced computation*, is a sequence

$$\mathcal{C} = (\gamma_0, r_1, \gamma_1, r_2, \ldots, r_m, \gamma_m, \ldots),$$

where $\gamma_0 \models A$ and, for all $i > 0$, $r_i$ is an **Ats**-action (a member of $\{\mathcal{R}_1,\ldots,\mathcal{R}_n\}$) such that $\gamma_{i-1} r_i \gamma_i$. The underlying sequence of states $\mathcal{C}_s$ is a *computation*, and the sequence of actions $\mathcal{C}_a$ is a *trace*. Assertion $A$ is the *initial condition*.

*Comments.* The union of a set of actions is an action, so the abstract transition system $(\Gamma, \{\mathcal{R}_1,\ldots,\mathcal{R}_n\})$ can be replaced by $(\Gamma, \{\mathcal{R}\})$, where $\mathcal{R} =_{def} \bigcup_{i=1}^n \mathcal{R}_i$, without changing the set of computations; a sequence $\mathcal{C}_s = (\gamma_n)$ is a computation if $\gamma_i$ is an $\mathcal{R}$-successor of $\gamma_{i-1}$, for all $i > 0$. A computation can be finite if it reaches a state without successor.

An **Ats**-*invariant*, or simply an *invariant*, is a predicate $I$ such that every successor of every state satisfying $I$ also satisfies $I$; this is denoted $\{I\}\,\mathcal{R}\,\{I\}$, or $\{I\}\,\mathbf{Ats}\,\{I\}$, or $\models (sp[I;\mathcal{R}] \Rightarrow I)$. An $(\mathbf{Ats}, A)$-*safety property*, or simply a *safety property*, is a predicate $J$ such that, for every computation $\mathcal{C}_s = (\gamma_0, \gamma_1, \ldots)$, if $\gamma_0 \models A$, then $\gamma_n \models J$ for all $n$.[1]

[1] In our framework, the connection between Hoare's logic and Dijkstra's calculus is simple: expressions $\{A\}\mathcal{S}\{B\}$, $\models (sp[A;\mathcal{S}] \Rightarrow B)$ and $\models (A \Rightarrow wlp[\mathcal{S};B])$ are equivalent. A useful property of *sp* (and *wlp*) is *monotonicity*. If $\mathcal{R}_1 \subseteq \mathcal{R}_2$ and $\models (A_1 \Rightarrow A_2)$, then $\models (sp[A_1;\mathcal{R}_1] \Rightarrow sp[A_2;\mathcal{R}_2])$. Similarly, if $\models (A_2 \Rightarrow A_1)$, $\mathcal{R}_2 \subseteq \mathcal{R}_1$ and $\models (B_1 \Rightarrow B_2)$, then $\{A_1\}\mathcal{R}_1\{B_1\}$ implies $\{A_2\}\mathcal{R}_2\{B_2\}$.

If $\models (A \Rightarrow I)$ and if $I$ is an invariant, then $I$ is necessarily a safety property, but it should be emphasized that the converse is not true: safety properties usually are not invariants. For instance, if **Ats** is a correct mutual exclusion algorithm, the assertion $J$ which expresses mutual exclusion is a safety property but is not an invariant.

*Comment.* With the restrictive definition given above, a computation $C$ can be checked for some safety property by considering only isolated states, and a safety property is simply (modelled by) a subset of $\Gamma$. Safety properties can be defined in a more general way [1] and modelled by subsets of $\Gamma^*$ ($\Gamma^*$ denotes the set of finite sequences of states). However, it is always possible, at least theoretically, to include all the preceding states in any state of the computation, so the restriction is not essential: any information about a computation prefix $(\gamma_0, \ldots, \gamma_n)$ can be retrieved from the state $\gamma_n$. In practice, special auxiliary variables, called *history variables*, are used for that purpose.

The following classical result (an early reference is [9]) asserts the completeness of the invariant method and states the connection between invariants and safety properties.

*Theorem.* The system $(\Gamma, \{\mathcal{R}\})$ satisfies the safety property $J$ for the initial condition $A$ if and only if an invariant $I$ exists such that $\models [(A \Rightarrow I) \wedge (I \Rightarrow J)]$.

*Sketch of proof.* The strongest possible choice for $I$ is $sin[A; \mathcal{R}]$, i.e., the set of states that can be accessed from $A$ (in finitely many computation steps). This predicate represents the set of $\mathcal{R}^*$-successors of all states satisfying $A$; it is an invariant, so $J$ is a safety property if and only if $\models (sin[A; \mathcal{R}] \Rightarrow J)$. □

*Comment.* Invariant are *inductive* safety properties, which can be proved by an induction argument. The standard technique for proving a (non-inductive) safety property is to construct a stronger, inductive one (i.e., an invariant). A similar situation frequently occurs in number theory. If some property $P(n)$ of natural numbers cannot be proved by induction, it is sometimes possible to discover a stronger property $Q(n)$ that can be proved by induction. Invariants are also named *stable properties*, e.g. in [6], where the word "invariant" refers to a stable property satisfied in some specified set of initial states.

### 2.3 Atomicity refinement: the abstract framework

Let $A$, $B$ be predicates on $\Gamma$, and let **Old** $= (\Gamma, \{\mathcal{S}, \mathcal{R}\})$, **New** $= (\Gamma, \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{R}\})$ be two abstract transition systems. Condition $A$ is the *initial condition*, and $B$ is the *refinement condition*. States satisfying $B$ are called *relevant states*; those not satisfying $B$ are *transient states*. We assume the following conditions:

1. $\models (A \Rightarrow B)$;
2. $\mathcal{S}_1$-successors of relevant states are transient states;
3. relevant states have no $\mathcal{S}_2$-successor;
4. $\mathcal{S}_2$-successors of transient states are relevant states;
5. transient states have no $\mathcal{S}_1$-successor;
6. $\mathcal{R}$-successors of relevant states are relevant states;
7. $\mathcal{R}$-successors of transient states are transient states;
8. $\mathcal{S} = \mathcal{S}_1; \mathcal{S}_2$ (sequential consistency).

These conditions[2] guarantee that, in any **New**-trace, actions $\mathcal{S}_1$ and $\mathcal{S}_2$ appear strictly in turn, and that $\mathcal{S}_1$ appears first. Predicate $A$ is the initial condition of both **Old** and **New** (only computations whose initial state satisfies $A$ are of interest). Predicate $B$ is the *refinement condition*, which is true in relevant states and false in transient states. Let $\mathcal{C} = (\gamma_0, r_1, \gamma_1, r_2, \ldots, r_m, \gamma_m, \ldots)$ be a **New**-traced computation (so $r_i \in \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{R}\}$, for all $i$). A state $\gamma_k$ is relevant if $\mathcal{S}_1$ and $\mathcal{S}_2$ occur equally many times in the trace prefix $\mathcal{P} = (r_1, \ldots, r_k)$; otherwise, $\gamma_k$ is transient (and $\mathcal{S}_1$ occurs one more time than $\mathcal{S}_2$ in $\mathcal{P}$).

*Comments.* We assume the existence of an atomicity refinement condition $B$. The simplest and most frequent case of atomicity refinement is the replacement of a transition $(\ell_0, S, \ell_1)$ by $(\ell_0, S_1, m)$ and $(m, S_2, \ell_1)$, where $S_1; S_2$ is "sequentially equivalent" to $S$ and where $m$ is a new label. The natural choice for the refinement condition is $B =_{def} \neg at\ m$ (the control does not lie at control point $m$, between $S_1$ and $S_2$). However, we also require that **Old** and **New** share the same state space $\Gamma$, and therefore the same assertion language. To ensure this, we assume that the new location predicate *at m* already existed in the old assertion language, even if no state satisfying it could be reached. Any assertion $J$ about **Old**, in particular the initial condition and the invariant, will be (maybe implicitly) rewritten as $J \wedge \neg at\ m$.

A **New**-trace is *primary* if every occurrence of $\mathcal{S}_1$ is immediately followed by an occurrence of $\mathcal{S}_2$. For most practical purposes, primary **New**-traces can be assimilated to **Old**-traces. The idea underlying trace reduction theorems is that, provided some hypotheses are satisfied, every **New**-trace has an equivalent **New**-primary trace, so **New** itself is equivalent to **Old**. The problem is, the stronger the equivalence notion, the stronger the required hypotheses. As a result, several trace reduction theorems have been proposed, with more or less restrictive hypotheses and equivalence notions.

## 2.4 Theorems

The trace reduction method allows to assert that some properties of **Old**-computations are preserved in **New**-computations. Even with restricting to safety properties, one cannot hope that all of them are preserved. For instance, with the notation of § 2.3, the refinement condition $B$ is an (**Old**,$A$)-safety property (and also an **Old**-invariant) but cannot be a (**New**,$A$)-safety property since $B$ is false in any transient state. However, if some hypothesis is satisfied, any **Old**-safety property $J$ gives rise to the **New**-safety property $B \Rightarrow J$. Otherwise stated, safety properties are preserved in relevant states, but nothing is known about transient states. Such a result is useful when $J$ is trivially true in transient states, i.e., when $\neg B \Rightarrow J$ is valid. This is a very frequent case; for instance, 2-process mutual exclusion and partial correctness are expressed by assertions that trivially hold in transient states, since critical states and final states (if any) always are relevant states.

The preservation theorem for safety property is an old result, originating from the ideas of [18] and [26]. The first formal presentation is probably [12]; [19] and [23] contain more results about atomicity refinement and the trace reduction method.

[2] Conditions 2 to 8 can be expressed as $\{B\}\,\mathcal{S}_1\,\{\neg B\}$, $\{B\}\,\mathcal{S}_2\,\{false\}$, $\{\neg B\}\,\mathcal{S}_2\,\{B\}$, $\{\neg B\}\,\mathcal{S}_1\,\{false\}$, $\{B\}\,\mathcal{R}\,\{B\}$, $\{\neg B\}\,\mathcal{R}\,\{\neg B\}$, and $sp[X;\mathcal{S}] \equiv sp[sp[X;\mathcal{S}_1];\mathcal{S}_2]$ for all $X$, respectively. Two useful corollaries are $\{true\}\,\mathcal{S}_1\,\{\neg B\}$ and $\{true\}\,\mathcal{S}_2\,\{B\}$.

A definition is introduced first:

**Definition.** A relation $\mathcal{R}_1$ *right-commutes* with a relation $\mathcal{R}_2$ (and relation $\mathcal{R}_2$ *left-commutes* with relation $\mathcal{R}_1$) if $\mathcal{R}_1;\mathcal{R}_2 \subseteq \mathcal{R}_2;\mathcal{R}_1$.

**Theorem 1.** If **Old**, **New**, $A$ and $B$ are as introduced in § 2.3, if $J$ is a predicate on $\Gamma$ and if $\mathcal{S}_1$ right-commutes with $\mathcal{R}$, then $B \Rightarrow J$ is a (**New**, $A$)-safety property if and only if $J$ is an (**Old**, $A$)-safety property.

**Proof of theorem 1.** The "only if" part is trivial. A direct proof of the "if" part is given in [12] and [23]; it is also a corollary of theorem 2 given below. □

*Comment.* Theorem 1 has a dual version, where requirement $\mathcal{S}_1$ right-commutes with $\mathcal{R}$ is replaced by $\mathcal{S}_2$ left-commutes with $\mathcal{R}$.

In order to compare the trace reduction technique and the invariant adaptation technique, we specify the connection between **Old**-invariants and **New**-invariants, when the reduction hypothesis holds.

**Theorem 2.** If **Old**, **New** and $B$ are as introduced in § 2.3, if $I$ is a predicate on $\Gamma$ such that $\models (I \Rightarrow B)$, and if $\mathcal{S}_1;\mathcal{R} \subseteq \mathcal{R};\mathcal{S}_1$ (that is, $\mathcal{S}_1$ right-commutes with $\mathcal{R}$), then predicate $I \vee sp[I;\mathcal{S}_1]$ is a **New**-invariant if and only if $I$ is an **Old**-invariant.

**Proof of theorem 2.** Let $\Phi$ be the predicate $I \vee sp[I;\mathcal{S}_1]$. We first assume that $\Phi$ is a **New**-invariant, and observe that $\Phi \wedge B$ is $I$. (Indeed, formula $\Phi \wedge B$ reduces to $(I \vee sp[I;\mathcal{S}_1]) \wedge B$, i.e., to $(I \wedge B) \vee (sp[I;\mathcal{S}_1] \wedge B)$, and the second disjunct is identically false.) From $\{\Phi\}\,\mathcal{R}\,\{\Phi\}$ and $\{B\}\,\mathcal{R}\,\{B\}$, we therefore deduce $\{I\}\,\mathcal{R}\,\{I\}$; from $\{\Phi\}\,\mathcal{S}_1\,\{\Phi\}$, $\{\Phi\}\,\mathcal{S}_2\,\{\Phi\}$ and $\{true\}\,\mathcal{S}_2\,\{B\}$ we deduce $\{\Phi\}\,\mathcal{S}_1;\mathcal{S}_2\,\{\Phi\}$ and $\{B\}\,\mathcal{S}_1;\mathcal{S}_2\,\{B\}$, and then $\{I\}\,\mathcal{S}_1;\mathcal{S}_2\,\{I\}$, therefore $\{I\}\,\mathcal{S}\,\{I\}$. As $\{I\}\,\mathcal{R}\,\{I\}$ and $\{I\}\,\mathcal{S}\,\{I\}$ both hold, $I$ is an **Old**-invariant.

We now assume that $I$ is an **Old**-invariant. In order to prove that $\Phi$ is a **New**-invariant, we check separately the triples $\{\Phi\}\,\mathcal{S}_1\,\{\Phi\}$, $\{\Phi\}\,\mathcal{S}_2\,\{\Phi\}$ and $\{\Phi\}\,\mathcal{R}\,\{\Phi\}$.

1. From the triples $\{I\}\,\mathcal{S}_1\,\{sp[I;\mathcal{S}_1]\}$ and $\{sp[I;\mathcal{S}_1]\}\,\mathcal{S}_1\,\{false\}$, we deduce $\underline{\{I \vee sp[I;\mathcal{S}_1]\}\,\mathcal{S}_1\,\{sp[I;\mathcal{S}_1] \vee false\}}$
2. $\{B\}\,\mathcal{S}_2\,\{false\}$ and $\{sp[I;\mathcal{S}_1]\}\,\mathcal{S}_2\,\{I\}$ lead to $\underline{\{B \vee sp[I;\mathcal{S}_1]\}\,\mathcal{S}_2\,\{I \vee false\}}$
3. Since $sp$ is monotonic, we get from the reduction hypothesis $\mathcal{S}_1;\mathcal{R} \subseteq \mathcal{R};\mathcal{S}_1$ $sp[I;(\mathcal{S}_1;\mathcal{R})] \Rightarrow sp[I;(\mathcal{R};\mathcal{S}_1)]$, i.e., $\{sp[I;\mathcal{S}_1]\}\,\mathcal{R}\,\{sp[sp[I;\mathcal{R}];\mathcal{S}_1]\}$. We have also $\{I\}\,\mathcal{R}\,\{I\}$, hence $\underline{\{I \vee sp[I;\mathcal{S}_1]\}\,\mathcal{R}\,\{I \vee sp[sp[I;\mathcal{R}];\mathcal{S}_1]\}}$

In every case the precondition is weaker than $\Phi$ and the postcondition is stronger, so the three required triples follow by monotonicity. (For the third postcondition, observe that $sp[I;\mathcal{R}] \Rightarrow I$, hence $sp[sp[I;\mathcal{R}];\mathcal{S}_1] \Rightarrow sp[I;\mathcal{S}_1]$.)

*Comment.* Let $\Psi$ be the strongest **New**-invariant which is implied by $I$, that is, the predicate $sin[I;(\mathcal{R} \cup \mathcal{S}_1 \cup \mathcal{S}_2)]$. A state $\gamma$ satisfies $\Psi$ if and only if there exists a **New**-computation $(\gamma_n : n = 0, 1, \ldots)$ such that $\gamma_0 \models I$ and $\gamma_k = \gamma$ for some $k \geq 0$. As $\Phi$ is a **New**-invariant implied by $I$, we have $\models (\Psi \Rightarrow \Phi)$; besides, $\models (\Phi \Rightarrow \Psi)$ also holds, since any state $\gamma$ satisfying $\Phi$ can be chosen as an initial state of computation (if $\gamma \models I$) or reached in a single step (if $\gamma \models sp[I;\mathcal{S}_1]$). This gives an interesting operational interpretation to the reduction hypothesis: every reachable transient state can be reached from some relevant state in exactly one step.

*Comment.* Here is the dual version of theorem 2. If **Old**, **New** and $B$ are as introduced above, if $I$ is a predicate on $\Gamma$ such that $\models (I \Rightarrow B)$, and if $\mathcal{S}_2$ left-commutes with $\mathcal{R}$,

then predicate $I \vee wlp[\mathcal{S}_2; I]$ is a **New**-invariant if and only if $I$ is an **Old**-invariant. The operator *wlp* (weakest liberal precondition) is defined as follows: $\gamma \models wlp[\mathcal{R}; J]$ if and only if every $\mathcal{R}$-successor of $\gamma$ satisfies $J$. Although the computation of $wlp[\mathcal{S}_2; I]$ can be easier than the computation of $sp[I; \mathcal{S}_1]$, we prefer to use the latter, which leads to a stronger **New**-invariant; as a program invariant is a formal description of its behaviour, the stronger is usually the better.

We can now show that, when the reduction hypothesis holds, the connection between the safety properties of **Old** and **New** is a mere consequence of the connection between the invariants of **Old** and **New**.

**Proposition.** The "if" part of theorem 1 is a corollary of theorem 2.[3]

**Proof.** If $J$ is an $(\mathbf{Old}, A)$-safety property, then, due to the completeness of the invariant method, there exists an **Old**-invariant $I$ such that $\models (A \Rightarrow I)$ and $\models (I \Rightarrow (B \wedge J))$.[4] If $\mathcal{S}_1$ right-commutes with $\mathcal{R}$ then (theorem 2), $\Phi =_{def} (I \vee sp[I; \mathcal{S}_1])$ is a **New**-invariant. Besides, it is easy to check[5] $\models (I \Rightarrow \Phi)$, $\models (A \Rightarrow \Phi)$, and $\models (\Phi \Rightarrow (B \Rightarrow J))$; as a result $B \Rightarrow J$ is a logical consequence of an (initially true) invariant, and therefore a $(\mathbf{New}, A)$-safety property. □

*Comment.* The fact $\models (\Phi \Rightarrow (B \Rightarrow J))$ will be useful later.

## 3 Trace reduction technique vs. invariant adaptation

In paragraph 2.4, the invariant adaptation method has been used to *justify* the trace reduction method. In this section, we would like to show that the invariant adaptation method can *replace* the trace reduction method. We will first show that, when an atomicity refinement can be validated by the trace reduction method, it can as easily be validated by the invariant adaptation method. Afterwards, we show that validation by invariant adaptation may happen to be tractable even when the reduction hypothesis is not satisfied.

### 3.1 The easy case of atomicity refinement

The data of the atomicity refinement problem are $\Gamma$, $\mathcal{S}$, $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{R}$, **Old**, **New**, $A$ and $B$, satisfying the 8 conditions stated in paragraph 2.3. Furthermore, we suppose that $J$ is an $(\mathbf{Old}, A)$-safety property, validated by an **Old**-invariant $I$. The question is to determine whether $B \Rightarrow J$ is a $(\mathbf{New}, A)$-safety property.

If we use the trace reduction technique, we have to verify that the reduction hypothesis $\mathcal{S}_1; \mathcal{R} \subseteq \mathcal{R}; \mathcal{S}_1$ holds. Theorem 2 asserts that a byproduct of this verification is the fact that $\Phi =_{def} (I \vee sp[I; \mathcal{S}_1])$ is a **New**-invariant. This fact alone is sufficient to validate the refinement (last comment of § 2.3). So, instead of checking whether the reduction hypothesis holds, we can check whether $\Phi$ is a **New**-invariant. In fact, we can do a bit less, as indicated by the next theorem.

**Theorem 3.** The assertion $\Phi$ is a **New**-invariant if and only if the assertion $sp[I; \mathcal{S}_1]$ is $\mathcal{R}$-invariant, i.e., if the triple $\{sp[I; \mathcal{S}_1]\}\, \mathcal{R}\, \{sp[I; \mathcal{S}_1]\}$ holds.

---

[3] Recall that the "only if" part of theorem 1 is trivial.

[4] Recall that $B$ characterizes relevant states, and therefore is a safety property of **Old**; transient states appear only in **New**-computations.

[5] Just consider separately the cases where $B$ is true and where $B$ is false; indeed, $\Phi$ can also be written as $(B \Rightarrow I) \wedge (\neg B \Rightarrow sp[I; \mathcal{S}_1])$.

**Proof.** Let us recall first that the assertion $\Phi$ reduces to $I$ when $B$ holds (relevant states) and to $sp[I;\mathcal{S}_1]$ when $\neg B$ holds (transient states). As a result, $\Phi$ is a **New**-invariant if and only if the following triples hold:

$1.\ \{I\}\mathcal{R}\{I\},\ 2.\ \{I\}\mathcal{S}_1\{sp[I;\mathcal{S}_1]\},\ 3.\ \{sp[I;\mathcal{S}_1]\}\mathcal{S}_2\{I\},\ 4.\ \{sp[I;\mathcal{S}_1]\}\mathcal{R}\{sp[I;\mathcal{S}_1]\}.$

Triple 2 is a tautology and triples 1 and 3 express that $I$ is an **Old**-invariant, so with this hypothesis triple 4 holds if and only $\Phi$ is a **New**-invariant. □

*Comment.* Validity of triple 4 is a weaker condition than the reduction hypothesis (theorem 2); furthermore, its verification can be easier. Indeed, the reduction hypothesis holds if and only if the implication

$$sp[P;\ (\mathcal{S}_1;\mathcal{R})]\ \Rightarrow\ sp[P;\ (\mathcal{R};\mathcal{S}_1)]$$

holds for each assertion $P$, whereas triple 4 can be rewritten in

$$sp[I;\ (\mathcal{S}_1;\mathcal{R})]\ \Rightarrow\ sp[I;\ \mathcal{S}_1],$$

i.e., an implication that must be true only for one specific assertion.

The conclusion is, when the trace reduction technique applies, the invariant adaptation technique also applies, with no more verification work.

### 3.2 The general case of atomicity refinement

The trace reduction technique might fail to validate a correct atomicity refinement, since this technique takes all states into account, even unreachable ones. (A notion of *context* has been introduced in [2] to deal with this problem.)

However, the invariant method might be useful even when theorem 2 does not apply. To investigate this, we have the following general theorem, which can be seen as a completeness theorem for atomicity refinement. It states that an atomicity refinement is correct if and only if some formula is an invariant.

**Theorem 4.** If **Old**, **New** and $B$ are as introduced above, and if $I$ is an **Old**-invariant such that $\models (I \Rightarrow B)$, then $B \Rightarrow I$ is a $(\mathbf{New}, I)$-safety property if and only if formula $\Phi^* =_{def} (I \vee sp[I;(\mathcal{S}_1;\mathcal{R}^*)])$ is a **New**-invariant.

*Comment.* Even when $B \Rightarrow I$ is a $(\mathbf{New}, I)$-safety property, it is usually not inductive; it is therefore not a **New**-invariant, but only the logical consequence of some **New**-invariant.

*Comment.* If $\mathcal{S}_1;\mathcal{R} \subseteq \mathcal{R};\mathcal{S}_1$, then formula $\Phi^*$ reduces to $\Phi =_{def} (I \vee sp[I;\mathcal{S}_1])$.

**Proof of theorem 4.** If $B \Rightarrow I$ is a $(\mathbf{New}, I)$-safety property, then any reachable relevant state satisfies $I$. Let $\gamma$ be a reachable transient state; there exist $n \geq 0$ and a traced computation prefix

$$\mathcal{C} =_{def} (\gamma_0, \mathcal{S}_1, \gamma_1, \mathcal{R}, \ldots, \gamma_i, \mathcal{R}, \ldots, \mathcal{R}, \gamma_{n+1})$$

such that $\gamma_0 \models I$ and $\gamma_{n+1} = \gamma$. As a result, $\gamma \models sp[I;(\mathcal{S}_1;\mathcal{R}^n)]$ and therefore $\gamma \models \Phi^*$. Any reachable state satisfies $\Phi^*$ and, clearly, any state satisfying $\Phi^*$ is reachable; so $\Phi^*$ is the set of reachable states, and therefore an invariant.

Conversely, if $\Phi^*$ is an invariant, it is also the set of reachable states, so all relevant reachable states satisfy $\Phi^* \wedge B$, that reduces to $I$. □

Theorem 4 can be the basis of a complete technique for validating atomicity refinements, but the problem is, computing $sp[I;(\mathcal{S}_1;\mathcal{R}^*)]$ is not easy in general.

We can now outline a more general comparison between trace reduction and invariant adaptation. Some notation is introduced first.

$$T_n =_{def} sp[I;(\mathcal{S}_1;\mathcal{R}^n)],$$
$$U_n =_{def} \bigvee_{i \leq n} T_i.$$
$$U^* =_{def} \bigvee_{i \geq 0} T_i.$$

The sequence $(U_n)$ is monotonic ($U_n \Rightarrow U_{n+1}$ holds for all $n$). An atomicity refinement is correct (theorem 4) if and only if $I \vee U^*$ is a **New**-invariant. A (correct) refinement is *stationary* if $U^*$ reduces to $U_n$ for some $n$. The preceding theorems imply that the trace reduction method works only if $U_0 = U^*$; even then, the notion of context introduced in [2] may be needed. The invariant-based technique is complete but, in practice, the computation of $U^*$ is likely to be intractable, except when $U^*$ reduces to $U_n$ for a small value of $n$. Three cases are of special interest:

1. $U^*$ reduces to $U_0$ and the trace reduction method does work.
2. $U^*$ reduces to $U_0$ and the trace reduction method does not work (except when contexts are used).
3. $U_1$ is weaker (i.e., greater) than $U_0$, and $U^*$ reduces to $U_1$; the trace reduction method does not work, but the invariant method remains tractable.

Case 2 is briefly illustrated in paragraph 4, where an example of case 3 is also mentioned.

### 3.3 Computer-aided verification

CAVEAT [16] is a tool for invariant validation. It also supports atomicity refinement, in so far only *sp*-calculus is used to produce invariant candidates $U_0$ and $U_1$. The practical bottleneck is that atomicity refinement induces quick size growing of the invariant, and therefore of the verification conditions. The general form of these conditions in CAVEAT is $(h_1 \ldots h_n) \Rightarrow c$, and the validation module becomes very slow when $n$ is big. A possible solution is to rank the hypotheses $h_1, \ldots, h_n$ according to their relevance to the conclusion $c$. Typically, very few hypotheses are really relevant, and even an elementary ranking program can speed up the validation process. Preliminary results are reported in [17].

## 4 Applications

When some requirements are satisfied, it is possible to solve (approximately) a fixpoint system of equations (e.g., on the domain of real numbers) like

$$\begin{cases} x = f(x,y) \\ y = g(x,y) \end{cases} \tag{1}$$

in a concurrent way, using two processes $X$ and $Y$ and two boolean variables $h_x$ and $h_y$, initialized to *true* [5, 11]. The processes are:

| *Process X* | *Process Y* |
|---|---|
| while $(h_x \vee h_y)$ do | while $(h_x \vee h_y)$ do |
| if $x \simeq f(x,y)$ | if $y \simeq g(x,y)$ |
| then $h_x := \mathit{false}$ | then $h_y := \mathit{false}$ |
| else $x := f(x,y)$; | else $y := g(x,y)$; |
| $(h_x, h_y) := (\mathit{true}, \mathit{true})$ | $(h_x, h_y) := (\mathit{true}, \mathit{true})$ |

(2)

The system terminates when both $h_x$ and $h_y$ are false; we would like that, on termination, both conditions $e_x =_{def} (x \simeq f(x,y))$ and $e_y =_{def} (y \simeq g(x,y))$ are satisfied.

In the coarser-grained version, there are only two transitions (and a single location for each process, say $X_0$ and $Y_0$ respectively). The transitions executed by process $X$ are

$$(X_0\,,\ (h_x \vee h_y) \wedge e_x \longrightarrow h_x := \mathit{false}\,,\ X_0)\,,$$
$$(X_0\,,\ (h_x \vee h_y) \wedge \neg e_x \longrightarrow (x, h_x, h_y) := (f(x,y), \mathit{true}, \mathit{true})\,,\ X_0)\,.$$

*Comment.* The relevant effect of the statement $x := f(x,y)$ is to assign unknown boolean values to *both* conditions $e_x$ and $e_y$.

An appropriate invariant of this coarse-grained version is $(h_x \vee e_x) \wedge (h_y \vee e_y)$. This formula is true initially (since $h_x$ and $h_y$ are both true) and respected by all transitions ($h_x$ and $h_y$ become false only when $e_x$ and $e_y$ are true, respectively, and every time $x$ or $y$ is touched, both variables $h_x$ and $h_y$ become true again). On termination, the invariant reduces to $e_x \wedge e_y$.

As a first atomicity refinement, we split the "else" part of process $X$, i.e., we replace

$$(X_0\,,\ (h_x \vee h_y) \wedge \neg e_x \longrightarrow (x, h_x, h_y) := (f(x,y), \mathit{true}, \mathit{true})\,,\ X_0)\,.$$

by

$$(X_0\,,\ (h_x \vee h_y) \wedge \neg e_x \longrightarrow x := f(x,y)\,,\ X_1)\,,$$
$$(X_1\,,\ (h_x, h_y) := (\mathit{true}, \mathit{true})\,,\ X_0)\,.$$

It is not possible to apply the reduction principle, since $x := f(x,y);\ y := g(x,y)$ and $y := g(x,y);\ x := f(x,y)$ may lead to distinct states; similarly, $(h_x, h_y) := (\mathit{true}, \mathit{true})$, in process $X$, and $h_y := \mathit{false}$ (in process $Y$) do not commute either. Nevertheless, the refinement is correct. To see this, we compute the first terms of the sequence $(T_n)$ introduced in paragraph 3.2.

The data are:

$I_0 : \mathit{at}\, X_0 \wedge \mathit{at}\, Y_0 \wedge (h_x \vee e_x) \wedge (h_y \vee e_y)$
$\mathcal{S}_1 : (X_0\,,\ (h_x \vee h_y) \wedge \neg e_x \longrightarrow x := f(x,y)\,,\ X_1)\,,$
$\mathcal{R}$: $\mathcal{R}_t \cup \mathcal{R}_f$, where
$\quad \mathcal{R}_t =_{def} (Y_0\,,\ (h_x \vee h_y) \wedge e_y \longrightarrow h_y := \mathit{false}\,,\ Y_0)\,,$
$\quad \mathcal{R}_f =_{def} (Y_0\,,\ (h_x \vee h_y) \wedge \neg e_y \longrightarrow (y, h_x, h_y) := (g(x,y), \mathit{true}, \mathit{true})\,,\ Y_0)\,.$

For $n = 0$, the disjunctive term $T_n =_{def} sp[I; (\mathcal{S}_1; \mathcal{R}^n)]$ reduces to $T_0 = sp[I_0; \mathcal{S}_1]$, i.e.

$$\mathit{at}\, X_1 \wedge \mathit{at}\, Y_0 \wedge h_x\,.$$

For $n = 1$, the disjunctive term $sp[I_0; (\mathcal{S}_1; \mathcal{R}^n)]$ reduces to $T_1 = sp[I_0; (\mathcal{S}_1; \mathcal{R})]$, and further to $sp[sp[I_0; \mathcal{S}_1]; \mathcal{R}_t] \vee sp[sp[I_0; \mathcal{S}_1]; \mathcal{R}_f]$, that is

$$\mathit{at}\, X_1 \wedge \mathit{at}\, Y_0 \wedge h_x \wedge [(e_y \wedge \neg h_y) \vee h_y]\,,$$

which further results in

$$\mathit{at}\, X_1 \wedge \mathit{at}\, Y_0 \wedge h_x \wedge (e_y \vee h_y)\,.$$

As $T_1$ is stronger then $T_0$, there is no need to compute further terms; $\bigvee T_n$ reduces to $T_0$. An acceptable invariant is now $I_1 =_{def} (I_0 \vee T_0)$, which can be simplified into

$$[(h_x \vee e_x) \wedge (h_y \vee e_y)] \vee (\mathit{at}\, X_1 \wedge h_x)\,.$$

This is an instance of case 2, since $U^*$ reduces to $U_0$

Symmetrically, if the "else" part of of process $Y$ is split, then the invariant is adapted into

$$[(h_x \vee e_x) \wedge (h_y \vee e_y)] \vee (at\, X_1 \wedge h_x) \vee (at\, Y_1 \wedge h_y).$$

A generalized version of algorithm (2) exists, which involves $n$ processes and allows the distributed solution of $n$-equation systems. However, the validation of atomicity refinements becomes more complicated, and involves several instances of case 3 (see [14] for details).

*Comment.* It should be emphasized that, for specific concurrent systems, easier validity proofs can be found for atomicity refinements. This paper is concerned only with the systematic techniques, applying to a broad class of concurrent systems.

## 5 Conclusion and related work

Two widely used methods for the validation of atomicity refinements have been compared. It is known for a long time that the invariant adaptation method is complete whereas the trace reduction method is not, but also assumed that, in some cases, the trace reduction method is easier to use. This assumption turns to be false and, as far as safety properties are concerned, the invariant-based method has definite advantages. Especially, many refinements encountered in classical examples are correct but outside the scope of the trace reduction techniques. Note, however, that the trace reduction method might still be useful to prove properties like termination and freeness of individual starvation; besides, other reduction methods (relying not only on traces) have been proposed.

The trace reduction technique has been successfully used especially in the area of (deterministic) parallel programming [2, 4]. The invariant adaptation technique is used e.g. in [10, 20]; a systematic presentation is [15]. Incremental construction of invariants, using approximation sequences like $(U_n)$, originates from [8, 7, 29]. Systematic approaches are [21] and [14].

Our main goal in this paper was to validate the decision made in CAVEAT, where the trace reduction method is not implemented (we plan to rely on invariant adaptation only). The program notation used in CAVEAT and in this paper is classical and allows for a convenient version of the reduction theorem and related results. From the theoretical point of view, however, these problems are better investigated at a more abstract, purely semantical level. An adequate framework for doing this is Lamport's TLA (Temporal Logic of Actions). In this formalism, both statements and assertions are represented as logical formulas; this leads to elegant and general formulations of results which, like the reduction theorem and other refinement theorems, involve more than one version of a program [22]. (TLA is also appropriate for more practical problems, especially in program specification; see [22, 25] for more details.) As pointed out by reviewers, the construction of the invariant of the refined version of a concurrent system in terms of the invariant of the reduced version can also be achieved in TLA, at a purely semantic level, as reported in an unpublished working paper [24]. The form given in the present paper (theorem 2) relies only on the elementary predicate transformer *sp*, and not on the higher-level predicate transformers *win* and *sin* used in [24], which cannot be implemented easily as such.

**Acknowledgment.** It is a pleasure to thank Yih-Kuen Tsay for improving the demonstration of theorem 2, and for a careful and critical reading of the manuscript.

## References


1. B. Alpern and F. Schneider, Recognizing safety and liveness, *Distributed Computing* **2** (1987) 117-126.
2. R.-J. Back, A Method for Refining Atomicity in Parallel Algorithms, *Lect. Notes in Comput. Sci.* **366** (1989) 199-216.
3. R.-J. Back and R. Kurki-Suonio, Decentralization of Process Nets with Centralized Control, *Distributed Computing* **3** (1989) 73-87.
4. R.-J. Back and R. Sere, Stepwise Refinement of Parallel Algorithms, *Sci. Comput. Programming* **13** (1990) 133-180.
5. E. Best, A Note on the Proof of a Concurrent Program, *Inform. Processing lett.* **9**, pp. 103-104, 1979
6. K.M. Chandy and J. Misra, *Parallel Program Design : A Foundation* (Addison-Wesley, Reading, MA, 1988).
7. E.M. Clarke, Synthesis of Resource Invariants for Concurrent Programs, *ACM Trans. Programming Languages Syst.* **2** (1980) 338-358.
8. P. Cousot and R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, *Proc. 4th ACM Symp. on Principles of Progr. Languages* (1977) 238-252.
9. J.W. De Bakker and L.G.L.T. Meertens, On the Completeness of the Inductive Assertion Method, *Jl. of Computer and Syst. Sci.* (1975) 323-357.
10. E.W. Dijkstra and al., On-the-Fly Garbage Collection : An Exercise in Cooperation, *Comm. ACM* **21** (1978) 966-975.
11. E.W. Dijkstra, Finding the Correctness Proof of a Concurrent Program, *Lect. Notes in Comput. Sci.* **69** (1979) 24-34.
12. T.W. Doeppner, Parallel Program Correctness Through Refinement, *Proc. 4th ACM Symp. on Principles of Progr. Languages* (1977) 155-169.
13. E.P. Gribomont, Synthesis of parallel programs invariants, *Lect. Notes in Comput. Sci.* **186** (1985) 325-338.
14. E.P. Gribomont, Stepwise refinement and concurrency : the finite-state case, *Sci. Comput. Programming* **14** (1990) 185-228.
15. E.P. Gribomont, Concurrency without toil : a systematic method for parallel program design, *Sci. Comput. Programming* **21** (1993) 1-56.
16. E.P. Gribomont and D. Rossetto, CAVEAT : technique and tool for Computer Aided VErification And Transformation, *Lect. Notes in Comp. Sci.* **939** (1995) 70-83.
17. E.P. Gribomont, Preprocessing for invariant validation, *submitted to AMAST'96.*
18. R.M. Keller, Formal Verification of Parallel Programs, *C. ACM* **19** (1976) 371-384.
19. Y.S. Kwong, On reduction of asynchronous systems, *Th. Comp. Sci.* **15** (1977) 25-50.
20. L. Lamport, An Assertional Correctness Proof of a Distributed Algorithm, *Sci. Comput. Programming* **2** (1983) 175-206.
21. L. Lamport, *win* and *sin* : Predicate Transformers for Concurrency, *ACM Trans. Programming Languages Syst.* **12** (1990) 396-428.
22. L. Lamport, The Temporal Logic of Actions, DEC SRC Report 79, 1989.
23. L. Lamport and F.B. Schneider, Pretending Atomicity, DEC SRC Report 44, 1989.
24. L. Lamport and F.B. Schneider, The Reduction Theorem, unpublished TLA note, available on `http://www.research.digital.com/SRC/tla/notes.html` , 1992.
25. L. Lamport and al., Introduction, papers and notes about TLA, available on `http://www.research.digital.com/SRC/tla/` .
26. R.J. Lipton, Reduction : A method of proving properties of parallel programs, *Comm. ACM* **18** (1975) 717-721.
27. G.L. Peterson, Myths about the mutual exclusion problem, *Information Proc. Lett.* **12** (1981) 115-116.
28. J. Sifakis, A unified approach for studying the properties of transition systems, *Theoret. Comput. Sci.* **18** (1982) 227-259.
29. A. van Lamsweerde and M. Sintzoff, Formal derivation of strongly correct concurrent programs, *Acta Inform.* **12** (1979) 1-31.