

Towards Model Checking Spatial Properties with SPIN^{*}

Alberto Lluch Lafuente

Department of Computer Science, Università di Pisa
lafuente@di.unipi.it

Abstract. We present an approach for the verification of spatial properties with SPIN. We first extend one of SPIN's main property specification mechanisms, i.e., the linear-time temporal logic LTL, with spatial connectives that allow us to restrict the reasoning of the behaviour of a system to some components of the system, only. For instance, one can express whether the system can reach a certain state from which a subset of processes can evolve alone until some property is fulfilled. We give a model checking algorithm for the logic and propose how SPIN can be minimally extended to include the algorithm. We also discuss potential improvements to mitigate the exponential complexity introduced by spatial connectives. Finally, we present some experiments that compare our SPIN extension with a spatial model checker for the π -calculus.

1 Introduction

SPIN [1] is a popular model checker used to verify temporal properties of concurrent systems. Part of its success is due to its efficiency. As a matter of fact, it is used as back-end model checker of various verification tools, like the first version of the Java Pathfinder [2], Bandera [3] or CheckVML [4]. SPIN can also be used [5, 6] to check properties of systems described in process algebras like the π -calculus [7].

System specifications in SPIN are given in PROMELA, a high-level language for defining communicating processes, while system properties can be given in a linear-time temporal logic (LTL) [8], in the more expressive formalism of Büchi automata, or by using other ad-hoc mechanisms to express deadlock freedom, satisfaction of local assertions, etc. In that manner SPIN is used to reason about temporal properties of concurrent systems. In some cases, however, one would like to restrict the reasoning to some parts of the system, only. For instance, in a client-server system one would like to express that, under some conditions, two clients can evolve together to reach some undesired state.

Such properties can be expressed by means of spatial logics, that are formalisms for expressing structural properties of models where there is a notion of

^{*} This work has been supported by the EU within the FETPI Global Computing, project IST-2005-016004 SENSORIA (*Software Engineering for Service-Oriented Overlay Computers*).

composition. Besides the usual boolean connectors, spatial logics include ingredients like the connective $\mathbf{0}$ to represent an empty model or the spatial decomposition $|$ which allows to write formulae $\phi_1|\phi_2$ that are satisfied by models that can be decomposed into two parallel submodels satisfying ϕ_1 and ϕ_2 , respectively.

The origins of logics to reason about temporal and spatial properties of systems can be tracked back on early work on logics for reasoning about networks of processes. For instance, the *multiprocess network logic* of [9] considers networks of interconnected processes and proposes a first-order logic with linear temporal modalities and spatial modalities *everywhere*, *anywhere* to respectively refer to the properties of *some* and *any* process connected to the process under consideration. This approach further inspired, amongst others, the logics ICTL* [10] and IPTL [11], *indexed* extensions of CTL* and LTL, respectively. The logic IPTL is actually the propositional fragment of the multiprocess network logic and ICTL* is an extension of CTL* (without the next-time operator) enriched with conjunctions and disjunctions of indexed propositions. In these early works the main focus was on reasoning about properties like mutual exclusion in systems with many identical processes.

More recently, several approaches to the verification of spatial properties have been proposed, on logics either for concurrent software system specifications given in process calculi like the π -calculus [12–14] or the ambient calculus [15], or data structures such as heaps [16], trees [17] and graphs [18]. Each such approach proposes a logic that combines spatial connectives with ingredients to reason about model specific aspects (names, childhood, edge adjacency, etc.).

Our goal is to extend the capabilities of SPIN in order to make it able to check spatial properties. We believe that this would be interesting both for system specifications given directly in PROMELA or in other formalisms used by formal methods tools (e.g. the ones cited above) that rely on SPIN for the verification. In most cases, structural aspects are a relevant issue that is worth specifying and verifying.

We achieve this by extending LTL with spatial connectives, much in the line of [14] which proposes a spatial version of a first-order μ -calculus to reason about spatial, temporal and name properties of π -calculus specifications. In contrast with [14] and the rest of the recent spatial logics cited above, we do not introduce the usual void and composition operators. Instead, we introduce second-order process variables that can be instantiated, compared or used to restrict the behavior that a formula refers to. More precisely, $\exists X.\phi$ expresses that ϕ holds for some subset X of the set of system processes, $[\phi]^X$ expresses that ϕ holds for runs restricted to transitions involving processes in set X , and formula $\psi_1 \subseteq \psi_2$ expresses that process expression ψ_1 is contained in process expression ψ_2 , where process expressions combine the empty set, singletons formed by process identities, set union and complementation and process set variables. The difference with the multiprocess network logic of [11] and similar works is that our process quantification is not used to reason about connections or local properties of processes but to refer to the behaviour to some processes only.

Our flavour of spatial connectives allows us to compactly express properties that, as far as we can see, are not expressible with the above cited approaches. For instance, the choice of a second-order quantifier has been inspired by the fact that the fixpoint-free fragment of the spatial logic of [18] is captured by Courcelle’s monadic second-order logic for graphs [19]. In our case, the usual spatial composition $\phi_1 \mid \phi_2$ can be seen as an abbreviation of $\exists X, Y. X \uplus Y = \mathcal{P} \wedge [\phi_1]^X \wedge [\phi_2]^Y$, where $[\phi]^X$ is the *strict* restriction (c.f. Section 2.3) of ϕ under X . It is worth noticing that the use of process quantification and restriction is strictly more expressive than the use of decomposition since the latter does not allow to reason about the whole system inside a decomposition. For instance, recalling the property of a client-server system suggested above we could not extend it to express whether in the undesired situation the server can always evolve to correct the situation. As a more concrete example, consider that one cannot express with the use of the decomposition operator that there is a set of processes X that lead the system to a state where there is another set of processes Y that satisfy a formula ϕ , a property that is expressed in our logic by $\exists X. [\diamond \exists Y. [\phi]^Y]^X$. We shall see that this is possible because existential quantification is not affected by restriction. In the formula above, for instance, $\exists Y$ quantifies over the whole set of processes of the system and not over X .

After giving the syntax and semantics of the spatial logic, we sketch a basic model checking algorithm and discuss its integration in SPIN. The algorithm is very simple and mainly relies on SPIN’s algorithm for LTL model checking. Indeed, we show that verification of the logic can be reduced to checking a certain set of LTL formulae. The extension of SPIN is minimal since it mainly requires us to add the capability of starting the search from a given state and restrict the set of active processes.

Next we face the complexity of the logic, which is strongly influenced by second-order quantification. Indeed, in order to check $\exists X. \phi$ we need to consider all subsets of the set \mathcal{P} of processes of the system, which number is exponential in the size of \mathcal{P} . This is a severe drawback on the performance of spatial model checkers. Analyzing all the possible subsets is, however, not always necessary. For instance, a typical property that is used by spatial model checkers is $\neg \mathbf{void} \wedge \neg(\neg \mathbf{void} \mid \neg \mathbf{void})$, abbreviated with $\mathbf{1}$, which expresses that a system has only one component. A property that can be clearly checked in constant time. Take now a formula $\mathbf{1} \mid \phi$, which expresses that a process can be removed such that the remaining system will satisfy ϕ . Clearly, only a linear number of decompositions have to be checked. While these cases can be dealt with by introducing explicit ingredients in the logic, the general problem remains open.

However, we argue that there is space for efficient algorithms. Consider a formula $\exists X. \phi$. Suppose we find that the system restricted to a set Q of processes does not satisfy ϕ . We know that, under some conditions, the system restricted to a superset P of Q simulates the system restricted to Q . For some formulae ϕ we can conclude that ϕ is neither satisfied by the system restricted to P without actually performing the verification. We shall see how to generalize some of these reasonings.

This paper is structured as follows. Section 2 presents the spatial logic for SPIN. Section 3 proposes a model checking algorithm and discusses methods to reduce the verification effort. Section 4 presents comparative empirical results with a spatial logic model checker. A last section concludes the paper and outlines current and future work.

2 Spatial Logic for SPIN

This section defines the syntax and semantics of our logic, which is interpreted over a suitable formalism. We also give some example properties of a well-known problem, namely Dijkstra’s dining philosophers problem, and suggestions for spatial property specification patterns.

2.1 Computational model

We introduce a variant of labeled transition systems as a suitable representation of the state space of PROMELA models over which our logic will be defined. We first recall the usual notion of (state) labeled transition systems.

Definition 1 (labeled transition system). A labeled transition system M is a tuple $\langle s_0, S, \rightarrow, L, AP \rangle$, where S is a set of states, $s_0 \in S$ is the initial state, $\rightarrow \subseteq S \times S$ is a transition relation and $L : S \rightarrow 2^{AP}$ is a labeling function mapping states into subsets of AP , a set of atomic predicates representing the observations on states.

Our notion of labeled transition considers that the system is composed by a set \mathcal{P} of processes.

Definition 2 (composed labeled transition system). A composed labeled transition system M is a tuple $\langle s_0, S, \rightarrow, L, AP, \mathcal{P} \rangle$, such that $\langle s_0, S, \rightarrow, L, AP \rangle$ is labeled transition system and \mathcal{P} is a finite set of processes that partitions the transition relation, i.e., $\rightarrow = \overset{\epsilon}{\rightarrow} \cup \bigcup_{p \in \mathcal{P}} \overset{p}{\rightarrow}$, where $\overset{\epsilon}{\rightarrow}$ is a set of empty transitions used for the usual stutter extension of finite runs¹.

Composed labeled transition systems roughly approximate PROMELA models without dynamic process creation where processes concurrently execute in an interleaving manner communicating via asynchronous operations². In the rest of the paper we shall use the term *labeled transition systems* or just *systems* to refer to *composed labeled transition systems*.

An *infinite run* of a system M at state s_0 , denoted M, s_0 is an infinite sequence $s_0 \rightarrow s_1 \rightarrow \dots$. A *maximal finite run* is a finite sequence $s_0 \rightarrow s_1 \rightarrow$

¹ We naively use transition labels above transitions and neglect the definition of labeled transitions as triples or via a labeling function to avoid confusion since only state labels play a role in the semantics.

² In PROMELA terminology, this mainly means that the `run` statement and synchronous channels are not allowed.

$\dots \rightarrow s_n$ such that there is no transition $s_n \rightarrow s_{n+1}$ in the system. A state s is called *ending state* if there is no transition departing from it. For the sake of simplicity, we assume that systems with finite maximal runs are represented by equivalent systems where the transition relation includes empty transitions that extend finite runs as usual, i.e., given a transition relation \rightarrow we define the *stutter extension* of \rightarrow by $se(\rightarrow) = \rightarrow \cup \{s \xrightarrow{\epsilon} s \mid \forall s \in S. (s, s') \notin \rightarrow\}$. In words, for each ending state s a self transition $s \xrightarrow{\epsilon} s$ is added. Thus, in the rest of the paper *run* denotes *infinite run*. extend finite runs as usual, i.e., given a transition relation \rightarrow we define the *stutter extension* of \rightarrow by $se(\rightarrow) = \rightarrow \cup \{s \xrightarrow{\epsilon} s \mid \forall s \in S. (s, s') \notin \rightarrow\}$. In words, for each ending state s a self transition $s \rightarrow_s$ is added. Thus, in the rest of the paper *run* denotes *infinite run*. The set of all runs of M starting at s is denoted by $\rho(M, s)$. For a run $r = s_0 \rightarrow s_1 \rightarrow \dots s_i \rightarrow s_{i+1} \rightarrow \dots$, the *suffix run of r starting at s_i* is $r^i = s_i \rightarrow s_{i+1} \rightarrow \dots$, while the i -th state of such a run r is denoted by s_i^r . A state $s \in S$ is *reachable* if there is a run r such that $s = s_i^r$ for some $i \in \mathbb{N}$.

Given a system M we define M_P as M restricted to its subprocesses $P \subseteq \mathcal{P}$, i.e., $M_P = \langle s_0, S, \rightarrow_P, L, AP, \mathcal{P} \rangle$. The restricted transition relation \rightarrow_P is defined by $se(\xrightarrow{\epsilon} \cup \bigcup_{p \in P} \xrightarrow{p})$, i.e., the stutter extension of the transitions of processes in P . Thus, the runs of M_P can be seen as the runs of M restricted to P , where a run r restricted to a set of processes P is denoted by r_P and is defined as the maximal prefix of r such that every transition $s \rightarrow s'$ belongs to some process in P , i.e., $s \rightarrow s' \in \bigcup_{p \in P} \xrightarrow{p}$. If no transition of r belongs to a process outside P then r_P is exactly r . Otherwise, r_P is like r until the first transition $s \rightarrow s'$ belonging to a process outside P is encountered. Then the rest of r_P is infinitely extended as explained above.

2.2 Logic syntax

Once we have a minimal formalism for a PROMELA model, we present the syntax of the spatial logic, where we assume a given system $M = \langle s_0, S, \rightarrow, L, AP, \mathcal{P} \rangle$.

Definition 3 (logic syntax). *Let V be a set of process variables. The syntax of our spatial logic is given by the following grammar:*

$$\begin{array}{ll}
\phi ::= T \mid \neg\phi \mid \phi \vee \phi \mid a & \text{(boolean connectives)} \\
\circ\phi \mid \phi \mathbf{U} \phi & \text{(temporal connectives)} \\
\psi \subseteq \psi \mid \exists X. \phi \mid [\phi]^X & \text{(spatial connectives)} \\
\psi ::= \emptyset \mid \{p\} \mid X \mid \psi \cup \psi \mid \bar{\psi} & \text{(set expressions)}
\end{array}$$

where $a \in AP$, $p \in \mathcal{P}$ and $X \in V$.

In the definition above V is assumed to contain a special variable \mathcal{X} that will be used to record the set of processes under which the system is restricted. We shall describe all ingredients in detail after the definition of the semantics of the logic. Here, we just advance the intuitive meaning of the spatial connectives and the set expressions. The rest of the ingredients constitute the well-known linear

temporal logic LTL [8], which we present here in a minimal manner by means of the *next-time* unary operator \circ and the *until* binary operator \mathbf{U} . Other typical operators can be derived from these, as we will recall in a next section.

Set connectives are nothing but the empty set, a singleton formed by the identity of a process, a set variable, set union and set complementation (with respect to \mathcal{P}). Set inclusion is thus trivially interpreted.

The second-order existential process quantifier binds X with a set of process which we shall see is a subset of the processes of the system. Then, a formula $[\phi]^X$ is valid if ϕ holds for the system restricted to processes in X . However, ϕ might contain further restriction operators which can change the set of process that restrict the system. In this way, we can reason about the behaviour of sub-components of a system without losing the capacity to refer to the behaviour of the whole system.

Before giving the formal semantics, we define the sets $fn(\phi)$ and $fn(\psi)$ of free variables of a formula ϕ and a set expression ψ . These are defined as expected.

Definition 4 (free variables). *Given a formula ϕ , the set $fn(\phi)$ of free variables of ϕ is defined by:*

$$\begin{aligned} fn(T) &= \emptyset \\ fn(\neg\phi) &= fn(\phi) \\ fn(\phi_1 \vee \phi_2) &= fn(\phi_1) \cup fn(\phi_2) \\ fn(a) &= \emptyset \\ fn(\circ\phi) &= fn(\phi) \\ fn(\phi_1 \mathbf{U} \phi_2) &= fn(\phi_1) \cup fn(\phi_2) \\ fn(\psi_1 \subseteq \psi_2) &= fn(\psi_1) \cup fn(\psi_2) \\ fn(\exists X.\phi) &= fn(\phi) \setminus \{X\} \\ fn([\phi]^X) &= fn(\phi) \cup \{X\} \end{aligned}$$

Similarly, given a set expression ψ , the set $fn(\psi)$ of free variables of ψ is defined by:

$$\begin{aligned} fn(\emptyset) &= \emptyset \\ fn(\{p\}) &= \emptyset \\ fn(X) &= \{X\} \\ fn(\psi_1 \cup \psi_2) &= fn(\psi_1) \cup fn(\psi_2) \\ fn(\overline{\psi}) &= fn(\psi) \end{aligned}$$

Obviously, the definition above assumes the usual safe renaming of variables such that variable names are not reused. We will consider closed formulae only, i.e., formulae where every process variable is bound by a quantifier, or in other words, formulae ϕ such that $fn(\phi) = \emptyset$. The notion of free names is crucial to define the equivalence axioms that we shall see in a next section.

2.3 Semantics

The semantics of our logic is interpreted over labeled transition systems.

Definition 5 (logic semantics). Let $M = \langle s_0, S, \rightarrow, L, AP, \mathcal{P} \rangle$ be a transition system, ϕ, ϕ_1, ϕ_2 be formulae, ψ, ψ_1, ψ_2 be set expressions, $X \in V$ be a second-order process variable, $\sigma : V \rightarrow 2^{\mathcal{P}}$ be a mapping of process variables into sets of processes, $s \in S$ and r be a run of M . The semantics of our logic is given by the following satisfaction relation:

$$\begin{aligned}
M, r \models_{\sigma} T &\Leftrightarrow \text{true} \\
M, r \models_{\sigma} \neg\phi &\Leftrightarrow M, r \not\models_{\sigma} \phi \\
M, r \models_{\sigma} \phi_1 \vee \phi_2 &\Leftrightarrow M, r \models_{\sigma} \phi_1 \text{ or } M, r \models_{\sigma} \phi_2 \\
M, r \models_{\sigma} a &\Leftrightarrow a \in L(s_0^r) \\
M, r \models_{\sigma} \circ\phi &\Leftrightarrow M, r^1 \models_{\sigma} \phi \\
M, r \models_{\sigma} \phi_1 \mathbf{U} \phi_2 &\Leftrightarrow \exists k \in \mathbb{N}. M, r^k \models_{\sigma} \phi_2 \text{ and } \forall 0 \leq j < k. M, r^j \models_{\sigma} \phi_1 \\
M, r \models_{\sigma} \psi_1 \subseteq \psi_2 &\Leftrightarrow \llbracket \psi_1 \rrbracket_{\sigma} \subseteq \llbracket \psi_2 \rrbracket_{\sigma} \\
M, r \models_{\sigma} \exists X. \phi &\Leftrightarrow \exists P \in 2^{\mathcal{P}}. M, r \models_{\sigma[P/X]} \phi \\
M, r \models_{\sigma} [\phi]^X &\Leftrightarrow M, s_0^r \models_{\sigma[\sigma(X)/\mathcal{X}]} \phi \\
M, s \models_{\sigma} \phi &\Leftrightarrow \forall r \in \rho(M_{\sigma(\mathcal{X})}, s). M, r \models_{\sigma} \phi
\end{aligned}$$

where $\llbracket \psi \rrbracket_{\sigma}$ is inductively defined by

$$\begin{aligned}
\llbracket \emptyset \rrbracket_{\sigma} &= \emptyset \\
\llbracket \{p\} \rrbracket_{\sigma} &= \{p\} \\
\llbracket X \rrbracket_{\sigma} &= \sigma(X) \\
\llbracket \psi_1 \cup \psi_2 \rrbracket_{\sigma} &= \llbracket \psi_1 \rrbracket_{\sigma} \cup \llbracket \psi_2 \rrbracket_{\sigma} \\
\llbracket \psi \rrbracket_{\sigma} &= \mathcal{P} \setminus \llbracket \psi \rrbracket_{\sigma}
\end{aligned}$$

Recall that \mathcal{X} is assumed to be a distinguished variable of V which we use to identify the set of processes under which the formula is restricted. Obviously, in the initial environment \mathcal{X} is mapped to the set \mathcal{P} of all processes of the system such that we say that M satisfies ϕ , written $M \models \phi$, whenever $M, s_0 \models_{\sigma[\mathcal{P}/\mathcal{X}]} \phi$ for any σ .

As the last equation of the satisfaction relation defined above suggests, a formula ϕ is satisfied by system M at state s if all runs of M restricted to $\sigma(\mathcal{X})$ starting at s satisfy ϕ . The satisfaction relation for boolean and temporal connectives is the usual one. Recall that $\circ\phi$ holds for a run r iff ϕ holds in the next state after the first state of r , while $\phi_1 \mathbf{U} \phi_2$ requires to ϕ_1 to hold until some point where ϕ_2 holds.

The inclusion of set expressions is defined as expected: $\psi_1 \subseteq \psi_2$ holds in environment σ whenever the set expression ϕ is included in the set expression ψ_2 , both under the environment σ . A formula $\exists X. \phi$ is satisfied by a run r in environment σ whenever there is a set of processes $P \subseteq \mathcal{P}$ for which ϕ holds for r in an environment σ' that is like σ except that variable X is mapped to P . Finally, $[\phi]^X$ holds for a run r of system M whenever all runs of M restricted to $\sigma(X)$ starting from the first state of r satisfy ϕ in a new environment where σ is updated to map \mathcal{X} to $\sigma(X)$.

In addition to the usual boolean and set abbreviations we enumerate the following ones:

$\diamond\phi \equiv \neg\Box\phi$	eventually
$\Box\phi \equiv \neg\diamond\neg\phi$	globally
$\phi_1\mathbf{R}\phi_2 \equiv \neg(\neg\phi_1\mathbf{U}\neg\phi_2)$	release
$\exists^1 X.\phi \equiv \exists X.\bigvee_{p\in\mathcal{P}} X = \{p\} \wedge \phi$	first-order quantifier
$\forall P.\phi \equiv \neg\exists X.\phi$	universal quantifier
$\lceil\phi\rceil^X \equiv \lceil\{\phi\}^X\rceil^X$	strict restriction
$\phi_1 \mid \phi_2 \equiv \exists X, Y. X \uplus Y = \mathcal{P} \wedge \lceil\phi_1\rceil^X \wedge \lceil\phi_2\rceil^Y$	composition
$\phi_1 \parallel \phi_2 \equiv \neg(\neg\phi_1 \mid \neg\phi_2)$	dual composition
$\mathbf{0} \equiv \mathcal{X} = \emptyset$	no process
$\mathbf{1} \equiv \neg\mathbf{0} \wedge (\mathbf{0} \parallel \mathbf{0})$	unique process

Here $\{\phi\}^X$ is formula ϕ relativized to X , which limits the reasoning in ϕ to processes inside X . This is mainly done by limiting every quantifier inside ϕ to subsets of X and intersecting every set expression with X .

Definition 6 (relativized formula). *A formula ϕ relativized to process variable X is defined by*

$$\begin{aligned}
\{T\}^X &= T \\
\{\neg\phi\}^X &= \neg\{\phi\}^X \\
\{\phi_1 \vee \phi_2\}^X &= \{\phi_1\}^X \vee \{\phi_2\}^X \\
\{a\}^X &= a \\
\{\circ\phi\}^X &= \circ\{\phi\}^X \\
\{\phi_1 \mathbf{U} \phi_2\}^X &= \{\phi_1\}^X \mathbf{U} \{\phi_2\}^X \\
\{\psi_1 \subseteq \psi_2\}^X &= (\psi_1 \cap X) \subseteq (\psi_2 \cap X) \\
\{\exists Y.\phi\}^X &= \exists Y. Y \subseteq X \wedge \{\phi\}^X \\
\{\lceil\phi\rceil^Y\}^X &= \exists Z. Z = Y \cap X \wedge \lceil\{\phi\}^X\rceil^Z
\end{aligned}$$

Most of the introduced abbreviations are typical in temporal and spatial logics. Regarding temporal abbreviations, we recall that $\diamond\phi$ is used to express that ϕ will eventually hold, $\Box\phi$ states that ϕ holds in every reachable state and $\phi_1\mathbf{R}\phi_2$ is the dual of the until operator, satisfied when ϕ_2 holds forever or until some point where both ϕ_1 and ϕ_2 hold. The spatial abbreviations can be used to express the number of processes present in the (sub)system or to reason about decompositions of the system. An interesting abbreviation is the strict restriction, which suggests the way one can force that a quantification that appears under a restriction is limited to the subsets of processes under which the formula is restricted. The strict restriction forbids a formula to refer to processes outside the subsystem under which it is restricted. The best example of use of the strict restriction are the composition operators. For instance, $\phi_1 \mid \phi_2$ holds whenever the processes of a system M can be decomposed into two (possibly empty) disjoint sets P, Q such that M_P, M_Q respectively satisfy ϕ_1, ϕ_2 under strict restriction. This means that ϕ_1 and ϕ_2 cannot refer to processes outside P and Q , respectively. Dually, $\phi_1 \parallel \phi_2$ holds whenever for all disjoint decompositions P, Q, M_P satisfies ϕ_1 or M_Q satisfies ϕ_2 , both under strict restriction.

2.4 Examples

We illustrate the use of our logic with Dijkstra’s dining philosophers problem [20]. Recall that it involves a number of philosophers sitting around a table. There is a plate in front of each philosopher and a fork between each pair of adjacent plates. A philosopher needs two forks to eat the spaghetti on his own plate. The problem is to find a protocol that allows the philosophers to use the forks in such a manner that they can all eat.

A strategy where every philosopher takes his left and right fork as a single atomic action is deadlock free but leads to starvation, i.e., philosophers are not guaranteed to eat infinitely often. On the other hand, consider a strategy where every philosopher decides to pick up his left fork and to not release it before a second fork has been acquired. It is clearly not deadlock free. It suffices to let each philosopher to pick up his left fork such that no philosopher will ever get its right fork.

We might refer to the state space of the problem as the system, where processes correspond to philosophers. In the following examples we will mainly make use of the compositional operator even if equivalent formulae written without such abbreviation are indeed simpler and more efficiently checked. The idea is to use a notation easy to understand to readers that are familiar with spatial logics and to explicit the fact that most of them are expressible in the spatial logic of [14], whose model checker we use in the the comparative experiments of a next section. However, we also give an example of a formula that is not expressible via the composition operator.

Assume that we want to reason about the second version of the problem. We know that it is not deadlock free, but we want to analyze the nature of deadlocks. Suppose that we have have an atomic proposition *deadlock* that holds in deadlock states only. A first spatial property that we might consider is that a deadlock occurs if and only if the system has no processes or all the processes collaborate. In other words, a strict subset of processes cannot lead the system into a deadlock state. This is expressed by the following formula:

$$\mathbf{prop1} \equiv (\diamond \mathit{deadlock}) \wedge (\neg \mathbf{0} \mid ((\diamond \mathit{deadlock}) \rightarrow \mathbf{0}))$$

Observe that $\mathbf{0}$ necessarily implies *deadlock*. Regarding our comment above, observe that the property could be more compactly and efficiently expressed by the formula $\neg \exists X. X \neq \emptyset \wedge X \neq \mathcal{P} \wedge [\diamond \mathit{deadlock}]^X$.

Now, we might wonder whether it is true that in every deadlock-free state of the protocol, it is possible to separate the philosophers in two groups such that one of the groups never deadlocks:

$$\mathbf{prop2} \equiv \Box(\neg \mathit{deadlock} \rightarrow ((\Box \neg \mathit{deadlock}) \mid T))$$

More generally, we might wonder whether in every deadlock-free state of the protocol, it is possible to separate the philosophers in two groups such that none of the groups ever deadlock:

$$\mathbf{prop3} \equiv \Box(\neg\mathit{deadlock} \rightarrow (\Box\neg\mathit{deadlock} \mid \Box\neg\mathit{deadlock}))$$

We also might want to consider properties about starvation. Assume that a proposition $p1\mathit{eats}$ holds in states where philosopher $p1$ is eating. The typical starvation-free property requires each philosopher to be able to eat infinitely often. In our case we can write $\Box\Diamond p1\mathit{eats}$ since by symmetry we know that it suffices to reason about one philosopher.

On the other hand, another interesting property could be whether it is possible, starting from a state in which $p1$ is not eating, to find a group of philosophers that can lead $p1$ into his eating state:

$$\mathbf{prop4} \equiv \Box(\neg p1\mathit{eats} \rightarrow (\Diamond p1\mathit{eats} \mid T))$$

Then, we can state a property requiring that, at any state where $p1$ is eating, we can find a group of philosophers that allows $p1$ to eat infinitely often:

$$\mathbf{prop5} \equiv \Box(p1\mathit{eats} \rightarrow (\Box\Diamond p1\mathit{eats} \mid T))$$

Finally, we might think of a property stating that a subset of processes can lead $p1$ into its eating state but then a set of processes can make him starve:

$$\mathbf{prop6} \equiv \exists X. [\Diamond(p1\mathit{eats} \wedge \exists Y. [\neg\Box\Diamond p1\mathit{eats}]^Y)]^X$$

It is worth noting that this property is not expressible by using the compositional abbreviation.

As an exercise we invite the reader to reason about the validity of the stated properties. The solution is given in Section 4.

2.5 Applications

We give further evidence of the practical interest of our logic by considering the extension of typical property specification patterns for finite-state verification. More precisely, we focus on the patterns identified in [21] (also available on the web³). There, patterns are mainly organized according to two categories: occurrence and order. The latter refer to the occurrence of a given event (e.g., absence, universality, existence), while the former regard the relative order in which multiple events occur (e.g., precedence, response, chains). In addition patterns, can be refined by limiting the scope where the property is intended to hold (globally, before or after an event, between two events, etc.). For the sake of simplicity, we will assume here a global scope.

Absence ($\Box\neg\phi$) and universality ($\Box\phi$) are similar patterns to state an invariant property represented by the (not necessarily atemporal) formula ϕ . Indeed, the only actual difference is the point of view of event ϕ : absence is typically

³ <http://patterns.projects.cis.ksu.edu/>

used to guarantee that some *undesired* event never happens and universality is used to ensure that a *good* property holds in every reachable state. A typical case where one could need spatial connectives is when it is known that the universality or absence of ϕ does not hold, but one is interested in identifying subsystems for which those properties still hold. This is what we did in the previous section regarding the absence of deadlocks.

Existence properties ($\diamond\phi$) express that some event will eventually happen. In the previous section we used this pattern to ensure the existence of a *bad* event, namely a deadlock and we combined it with spatial connectives to guarantee that only the whole system could make that bad event happen. Typical applications require ϕ to be a *good* event which does not happen if the system has cyclic behaviours where ϕ never happens. Hence, in some cases we could allow such behaviours but require a set of processes to exist that can exit the cyclic behaviour and ensure the existence of ϕ . This can be expressed with the formula $\Box(\neg\phi \rightarrow \exists X.[\diamond\exists Y = \mathcal{P}.[\diamond\phi]^Y]^X)$.

Combining the universality and existence patterns also results in the *infinitely often* pattern $\Box\diamond\phi$ that we used to state absence of starvation in the example of the previous section. By nesting spatial ingredients inside such a pattern we can write a formula $[\Box[\diamond\phi]^Y]^X$ that can be used to express that in any state reachable by processes of X the processes Y ensure occurrence of ϕ . This might be useful in case we have *bad* processes (X) trying to avoid the *good* event (ϕ) and *good* processes (Y) ensuring the occurrence of the good event.

Precedence of event ϕ_1 over ϕ_2 is expressed by $\neg\phi_1 \mathbf{W}\phi_2$, where $\phi_1 \mathbf{W}\phi_2$ abbreviates the *weak until* operator $\phi_1 \mathbf{U}(\phi_2 \vee \Box\phi_1)$. Response properties ($\Box(\phi_1 \rightarrow \diamond\phi_2)$) are used to require that every request ϕ_1 will be followed by a response ϕ_2 . In both cases we can use spatial connectives to restrict the reasoning to some processes only. For instance, we can strengthen a response property by requiring that a process X alone (e.g., a server) achieves to produce the response. This is stated by $\Box(\phi_1 \rightarrow [\diamond\phi_2]^X)$.

Chain patterns are used to express relations of complex combinations of events. These include precedence or response relationships consisting of sequences of individual events. We consider the 1-stimulus, 2-response pattern here, expressed by $\Box(\phi_1 \rightarrow \diamond(\phi_2 \wedge \circ\diamond\phi_3))$, where ϕ_1 is the request or stimulus and ϕ_2, ϕ_3 are the responses. The formula states that every event ϕ_1 is eventually followed by ϕ_2 and ϕ_3 (in this order). As proposed in the above paragraph, the occurrence of both events can be restricted to different sets of processes, i.e., $\Box(\phi_1 \rightarrow [\diamond(\phi_2 \wedge [\circ\diamond\phi_3]^Y])^X)$.

3 Model Checking

We face here the model checking problem for the spatial logic presented in the previous Section by sketching a basic algorithm and proposing potential improvements.

3.1 Basic Algorithm

The algorithm for checking spatial formulae mainly relies on SPIN's capacity to check LTL formulae. Recall that the mechanism used by SPIN in order to check LTL properties is the so-called *automata-based model checking* approach. Roughly speaking, in order to check that a system M satisfies a formula ϕ a Büchi automaton A is constructed as the intersection of the Büchi automata corresponding to M and $\neg\phi$. Intuitively, A accepts the infinite runs accepted by both the system and the negation of ϕ , i.e., it models the behaviours of the system violating property ϕ .

This is done in SPIN by implementing a Büchi automaton as a special process called *never claim* that is executed concurrently with the system. The executability of the transitions of the never claim depends on boolean expressions on the system variables that represent the atomic propositions of the corresponding formula.

```

Algorithm check(P,s,σ,φ) switch  $\phi$  do
  case  $\psi_1 \subseteq \psi_2$ 
    return set(ψ1,σ) ⊆ set(ψ2,σ) ;
  case  $\exists X.\phi_1$ 
    foreach  $P_1 \in 2^P$  do
       $\sigma[X] := P_1$ ;
      if spin(P,s,σ,φ1) then return true ;
    return false;
  case  $[\phi_1]^X$ 
    return spin(σ[X],s,σ,φ1) ;

```

Fig. 1. Procedure `check` for checking spatial subformulae.

Our algorithm exploits the fact that the transitions of the never claim are PROMELA statements which can include, for instance, calls to procedures written in C. The idea is that we convert spatial formulae into maximal LTL subformulae where spatial formulae are substituted by a call to a special verification procedure, which possibly relies on SPIN again. This approach has the benefit to require minimal changes in SPIN.

Thus, the first thing to do is to convert a spatial formula into a set of pure LTL formulae whose atomic propositions are related to spatial subformulae.

Definition 7. Let ϕ be a formula. The corresponding flat formula $\text{flat}(\phi)$ is defined by

$$\begin{aligned}
\text{flat}(T) &= T \\
\text{flat}(\neg\phi) &= \neg\text{flat}(\phi) \\
\text{flat}(\phi_1 \vee \phi_2) &= \text{flat}(\phi_1) \vee \text{flat}(\phi_2) \\
\text{flat}(a) &= a \\
\text{flat}(\circ\phi) &= \circ\text{flat}(\phi) \\
\text{flat}(\phi_1 \mathbf{U} \phi_2) &= \text{flat}(\phi_1) \mathbf{U} \text{flat}(\phi_2) \\
\text{flat}(\psi_1 \subseteq \psi_2) &= a_{\psi_1 \subseteq \psi_2} \\
\text{flat}(\exists X.\phi) &= a_{\exists X.\text{flat}(\phi)} \\
\text{flat}([\phi]^X) &= a_{[\text{flat}(\phi)]^X}
\end{aligned}$$

Thus, when converting the negation of a flattened formula into the corresponding never claim, each atomic proposition a_ϕ is actually replaced by a call to $\text{check}(P, s, \sigma, \phi)$. Procedure check is depicted in Figure 1. It takes four parameters as input and returns a boolean value. The first parameter is P which represents the set of processes under which the formula is restricted. The second parameter s is the system state from which the formula must be checked. The third parameter is an array σ of processes that ranges over variable names. It implements the environment of process variables. Finally, ϕ is a flat formula.

Observe that the procedure relies on an extension of SPIN which we refer to as spin . The extension is minimal and allows to restrict the execution to a subset of processes, start the verification from a given state and record the array σ .

Set expressions are trivially checked via a procedure set . Process quantification requires us to consider all possible subsets P_1 of \mathcal{P} . For each such set, $\sigma[X]$ is updated with its value and spin is called with the new value for σ in order to check ϕ . Finally, process restriction $[\phi_1]^X$ is checked by calling spin using the set of processes assigned to X as first parameter.

3.2 Spatial equivalences

The exponential complexity introduced by the second-order process quantification can be mitigated in some cases by rewriting the formula in an appropriate manner. For instance, a formulae $\exists X.\phi$ is trivially equivalent to ϕ if X is not a free variable of ϕ . As a first step towards such a simplification we introduce a set of structural axioms for spatial formulae which induces an equivalence of formulae.

Definition 8 (spatial equivalence). *The spatial equivalence is the least relation \equiv_s on formulae closed under the following axioms:*

$$\begin{array}{lll}
\exists X.T & \equiv_s & T \\
\exists X.\phi_1 \vee \phi_2 & \equiv_s & \phi_1 \vee \exists X.\phi_2 & \text{if } X \notin \text{fn}(\phi_1) \\
\exists X.a & \equiv_s & a \\
\exists X.\circ\phi & \equiv_s & \circ\exists X.\phi \\
\exists X.\phi_1 \mathbf{U}\phi_2 & \equiv_s & (\exists X.\phi_1) \mathbf{U}\phi_2 & \text{if } X \notin \text{fn}(\phi_2) \\
\exists X.\phi_1 \mathbf{U}\phi_2 & \equiv_s & \phi_1 \mathbf{U}\exists X.\phi_2 & \text{if } X \notin \text{fn}(\phi_1) \\
\exists X.\psi & \equiv_s & \psi & \text{if } X \notin \text{fn}(\psi) \\
\exists X.\exists Y.\phi & \equiv_s & \exists Y.\exists X.\phi \\
\exists X.[\phi]^Y & \equiv_s & [\exists X.\phi]^Y & \text{if } X \neq Y \\
[T]^X & \equiv_s & T \\
[\neg\phi]^X & \equiv_s & \neg[\phi]^X \\
[\phi_1 \vee \phi_2]^X & \equiv_s & \phi_1 \vee [\phi_2]^X & \text{if } X \notin \text{fn}(\phi_1) \text{ and } \phi_1 \text{ atemporal} \\
[a]^X & \equiv_s & a \\
[\psi_1 \subseteq \psi_2]^X & \equiv_s & \psi_1 \subseteq \psi_2 \\
[\exists Y.\phi]^X & \equiv_s & \exists Y.[\phi]^X
\end{array}$$

In the definition above a formula ϕ is *atemporal* if it does not contain any temporal operator. Next, we state that the satisfaction relation is closed under spatial equivalence.

Proposition 1. *Let ϕ_1, ϕ_2 be two spatial formulae and M be a transition system. If $\phi_1 \equiv_s \phi_2$ we have $M \models \phi_1$ whenever $M \models \phi_2$.*

Proof (sketch). The proof is trivial: one can basically apply the reasonings usual in propositional logics. The only case that is worth mentioning regards the axiom $\exists X.\circ\phi \equiv_s \circ\exists X.\phi$. But this is not a problem since we assume that the set of processes \mathcal{P} is constant.

In current work we are investigating heuristics for deciding, given a certain formula ϕ_1 , how to rewrite it into an equivalent formula ϕ_2 such that checking ϕ_2 requires significantly less effort, possibly by introducing additional axioms for typical abbreviations. For instance, consider formula $\Box\exists X.\Box\phi$. One can easily show that $\Box\Box\phi \equiv \Box\phi$ and $\Box\exists X.\phi \equiv \exists X.\Box\phi$ are equivalence axioms of our logic. This leads to the equivalent formula $\exists X.\Box\phi$ which clearly requires less verification effort with our basic algorithm.

3.3 Exploiting simulations

Checking $\exists X.\phi$ requires us in general to consider the $2^{\mathcal{P}}$ decompositions of the system. However, in some cases such a formula can be checked more efficiently. Consider for instance formula **1** expressing that there is just one process in the system, which can be checked in constant time, or formulae of the form $(\mathbf{1} \wedge \phi) \mid \psi$ stating that there is a decomposition of the system where one part consists of a single process satisfying ϕ and the rest satisfies ψ , which requires considering a

linear number of decompositions, only. Another typical example is the formula $\forall^1 X. [\phi]^X$ or, equivalently, $false \parallel (\mathbf{1} \rightarrow \phi)$ expressing that the system restricted under any single process satisfies ϕ .

A simple way to tackle the problem in specific cases is to explicitly include such abbreviations in the logic and implement ad-hoc procedures for them. However, it is worth studying procedures for the general case. For instance, one can try to exploit the fact that a system approximates the behaviour of its subsystems. We first define a well-known notion of approximation.

Definition 9 (simulation). *Given two transition systems $M_1 = \langle s_1^0, S_1, \rightarrow^1, L_1, AP_1, \mathcal{P}_1 \rangle$, $M_2 = \langle s_2^0, S_2, \rightarrow^2, L_2, AP_2, \mathcal{P}_2 \rangle$, a relation $R \subseteq S_1 \times S_2$ is a simulation relation whenever $s_1 R s_2$ implies:*

- $L(s_1) = L(s_2)$;
- for every transition $s_1 \rightarrow^1 s'_1$ there is a transition $s_2 \rightarrow^2 s'_2$ such that $s_2 R s'_2$.

If there is a simulation relation R such that $s_1 R s_2$ we say that M_2 at s_2 simulates M_1 at s_1 , written $M_1, s_1 \preceq M_2, s_2$. If s_1, s_2 are s_1^0, s_2^0 we just say that M_2 simulates M_1 , written $M_1 \preceq M_2$.

We next observe that under some conditions a subsystem M_P simulates its subsystem M_Q . That this is not true in general can be easily shown. The problem relies in the fact that M_Q might contain empty transitions that are not present in M_P and that might not be simulated. An example are deadlocks present in M_Q .

A sufficient condition for guaranteeing the that M_P satisfies M_Q is that M_P preserves the ending states of M_Q . This means that in the states where the processes in Q cannot progress, neither can the processes in $P \setminus Q$. A trivial but realistic case where this happens is when the processes in Q do not communicate with processes in $P \setminus Q$. Hence, if we have $Q \subseteq P \subseteq \mathcal{P}$ we say that P is a *fair superset* of Q whenever for any reachable ending state s in M_Q state s is also an ending state in M_P .

Proposition 2 (fair supersets simulate). *Let M be a transition system and P, Q be two sets of processes such that P is a fair superset of Q then, for any $s \in S$:*

$$M_Q, s \preceq M_P, s.$$

Proof (sketch). The identity relation $id = \{(s, s) \mid s \in S\}$ is clearly a simulation relation since both $L(s) = L(s)$ for any $s \in S$ and $\rightarrow_Q \subseteq \rightarrow_P$.

The notion of simulation is sufficient to preserve satisfaction of the temporal logic ACTL* and thus LTL [22] but not our full logic due to the spatial operators. However, we can still identify fragments of our spatial logic that are preserved by fair supersets of processes. The class of preserved formulae is characterized by a type system, which may assign to a formula ϕ the type “ \rightarrow ”, meaning that ϕ is preserved by fair supersets or the type “ \leftarrow ”, meaning that ϕ is reflected by fair supersets.

Definition 10 (preserved formulae). *The typing rules are given by*

$$\begin{array}{l}
T, a : \leftrightarrow \\
\neg\phi : d^{-1} \quad \text{if } \phi : d \\
\phi_1 \vee \phi_2 : d \quad \text{if } \phi_1 : d \text{ and } \phi_2 : d \\
\circ\phi : d \quad \text{if } \phi : d \\
\phi_1 \mathbf{U}\phi_2 : d \quad \text{if } \phi_1 : d \text{ and } \phi_2 : d \\
\psi_1 \subseteq \psi_2 : \rightarrow \quad \text{if } \psi_1 \subseteq \psi_2 \text{ antimonotonic} \\
\psi_1 \subseteq \psi_2 : \leftarrow \quad \text{if } \psi_1 \subseteq \psi_2 \text{ monotonic} \\
\exists X.\phi : d \quad \text{if } \phi : d \\
[\phi]^X : d \quad \text{if } \phi : d
\end{array}$$

where it is intended that $\rightarrow^{-1} = \leftarrow$ and $\leftarrow^{-1} = \rightarrow$. Moreover $\phi : \leftrightarrow$ is a shortcut for $\phi : \rightarrow$ and $\phi : \leftarrow$, while $\phi_1, \phi_2 : d$ stands for $\phi_1 : d$ and $\phi_2 : d$.

In the definition above $\psi_1 \subseteq \psi_2$ *antimonotonic* means that for any pairs of mappings σ_1, σ_2 that ensure that for any variable $X \in V$ it holds $\sigma_1(X) \subseteq \sigma_2(X)$ we have that $\llbracket \psi_1 \subseteq \psi_2 \rrbracket_{\sigma_2}$ implies $\llbracket \psi_1 \subseteq \psi_2 \rrbracket_{\sigma_1}$. Monotonicity of process set inclusions is defined similarly.

We now state that preserved (resp. reflected) formulae are indeed preserved (resp. reflected) by fair supersets.

Proposition 3 (preservation). *Let M be a transition system and P, Q be two sets of states such that P is a fair superset of Q , and let ϕ be a formula. Then for any $s \in S, X \in V$ we have*

$$\begin{array}{l}
M, s \models_{\sigma[P/X]} \phi \Rightarrow M, s \models_{\sigma[Q/X]} \phi \quad \text{if } \phi : \rightarrow \\
M, s \models_{\sigma[P/X]} \phi \Leftarrow M, s \models_{\sigma[Q/X]} \phi \quad \text{if } \phi : \leftarrow
\end{array}$$

Proof (sketch). Let us consider the first equation (the other is dual). Note that we have to prove that the satisfaction relation is antimonotonic with respect to any variable X but this is clear from the definition of preserved formulae. Regarding the temporal connectives observe that we have M_P simulates M_Q at s .

Proposition 3 gives an intuition on how to improve the check of spatial operators. If, for instance, we have to check whether a system M satisfies formula $\exists X.\phi$ with ϕ being a preserved formula and we find out that $M, s \not\models_{\sigma[Q/X]} \phi$ we know that $M, s \not\models_{\sigma[P/X]} \phi$ for any fair superset P of Q . Determining the fair supersets of Q can be based on a static analysis. This suggests the improved version of the check procedure depicted in Figure 2.

4 Experiments.

We compared a first implementation of the basic algorithm as an extension of the SPIN model checker with SLMC [23] a spatial model checker that implements the approach described in [14] for checking spatial properties of π -calculus specifications. The goal of the experiments is to have a first impression of the efficiency

```

Algorithm  $\text{check}(P, s, \sigma, \phi)$  switch  $\phi$  do
  case  $\psi_1 \subseteq \psi_2$ 
    return  $\text{set}(\psi_1, \sigma) \subseteq \text{set}(\psi_2, \sigma)$  ;
  case  $\exists X. \phi_1$ 
    foreach  $P_1 \in 2^{\mathcal{P}}$  do
       $\text{valid}[P_1] := \text{unknown}$  ;
    foreach  $P_1 \in 2^{\mathcal{P}}$  such that  $\text{valid}[P_1] \neq \text{false}$  do
       $\sigma[X] := P_1$ ;
      if  $\text{spin}(P, s, \sigma, \phi_1)$  then return true ;
      if preserved( $\phi$ ) then
        foreach  $P_2 \in 2^{\mathcal{P}}$  such that  $\text{fair}(P_2, P_1)$  do
           $\text{valid}[P_2] := \text{false}$  ;
    return false;
  case  $[\phi_1]^X$ 
    return spin( $\sigma[X], s, \sigma, \phi_1$ );

```

Fig. 2. Procedure `check` for checking preserved spatial subformulae exploiting supersets.

of our basic algorithm by comparing it with the only spatial model checker we are aware of.

Before presenting the results we reveal the solution for the validity of the properties described in Section 2.4. Recall that we consider a deadlock solution to Dijkstra’s dining philosophers problem consisting in a protocol for the philosophers such that they first try to catch the left fork and then the right one. A philosopher that manages to get both forks is considered to be in its eating state whose only outgoing transition corresponds to releasing both forks at the same time. Properties **prop1**, **prop2**, **prop5** and **prop6** hold while **prop3** and **prop4** do not hold.

Table 1 depicts the results. The second and third columns respectively correspond to the results of the spatial version of SPIN and the SLMC model checker. Each row presents the respective running time (in seconds) achieved when checking the properties described in section 2.4 in PROMELA and π -calculus models of the deadlock solution to Dijkstra’s dining philosophers problem. For each property we have tested instances of the problem with increasing number n of philosophers. We restrict the results to two instances per problem with a limit of 10 philosophers and a time limit of 30 minutes.

We are currently investigating the reasons why the performance of both model checkers drastically differ depending on the property checked. However, the fact that SPIN offers a better performance in some cases is encouraging because our current implementation is limited to the basic algorithm. Additional

Problem	Spatial SPIN	SLMC
prop1 , $n = 9$	0.56	3.60
prop1 , $n = 10$	1.19	16.11
prop2 , $n = 5$	5.08	0.45
prop2 , $n = 6$	21.58	1.71
prop3 , $n = 8$	0.13	32.41
prop3 , $n = 9$	0.32	240.87
prop4 , $n = 3$	0.04	0.14
prop4 , $n = 4$	0.10	o.t.
prop5 , $n = 5$	26.30	0.26
prop5 , $n = 6$	740.37	1.08

Table 1. Experimental results

experiments show that SPIN is much more efficient for exhaustive exploration of the model as well as for deadlock detection.

5 Conclusion

We have proposed a spatial logic for the SPIN model checker. The logic basically extends LTL with spatial connectors to quantify over the set of component processes of a system and restrict a formula under such sets. In such a manner we can express properties of that relate the behaviour of the components of the system.

We have sketched a basic implementation that requires minimal changes in the implementation of SPIN. Next, we have discussed possible ways to reduce the complexity introduced by the second order process quantifier. First, we defined a set of structural axioms under which the satisfaction of spatial formula is closed. Then, we identified a fragment of the logic that is preserved by some *fair* subsystems, such that if we find out that such a formula does not hold for the system restricted to the set of processes Q we can neglect checking ϕ for the fair supersets of Q .

Last, we presented a set of comparative experiments with the spatial model checker SLMC [23] which show that our basic algorithm offers a better performance in some cases. This encourages us to implement the proposed improvements and further study heuristics for an efficient verification of our logic.

In current work we are implementing our whole approach, analyzing its complexity, investigating ways to rewrite formulae into equivalent ones that are more easy to check, and studying trade-offs between the relaxation of the notion of *fair* supersets and the reduction of preserved fragments in order to achieve good performance. A good starting point could be to exploit classical results of compositional reasoning [24] and verification of infinite families of finite-state systems [22]. Potential directions of future research consists of extending the approach to systems with dynamic creation of processes or including first-order

predicates in the line of [9] such that also state observations are affected by process restriction.

We also plan to analyze the application of state space reduction techniques in our approach, with a special focus in those techniques implemented in SPIN and in extensions of it. For example, we would like to show whether partial order reduction [25] can be soundly applicable and investigate how to exploit the fact that in the calls to SPIN some of the processes are inactive which means that some interferences of transitions in the whole system can be guaranteed to be absent in some subsystems. As another example, when the focus is on bug finding, the *directed model checking* approach [26] can be used to accelerate the search for errors and possibly provide shorter counterexamples. One can study heuristics to decide the order in which to consider the assignment of subsets to a variable such that errors are found faster.

References

1. Holzmann, G.: The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, Reading, Massachusetts (2004)
2. Havelund, K., Pressburger, T.: Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer* **2**(4) (2000)
3. Corbett, J.C., Dwyer, M.B., Hatcliff, J., Laubach, S., Pasareanu, C.S., Robby, Zheng, H.: Bandera: Extracting finite-state models from Java source code. In: 22nd International Conference on Software Engineering (ICSE), IEEE Computer Society (2000)
4. Varró, D.: Automated formal verification of visual modeling languages by model checking. *Software and System Modeling* **3**(2) (2004) 85–113
5. Song, H., Compton, K.J.: Verifying π -calculus processes by promela translation. Technical Report CSE-TR-472-03, University of Michigan (2003)
6. Wu, P.: Interpreting π -calculus with spin/promela. Technical report, Lab. for Computer Science, Institute of Software, Chinese Academy of Sciences (2001)
7. Milner, R.: Communicating and Mobile Systems: The π -calculus. Cambridge University Press (1992)
8. Manna, Z., Pnueli, A.: The temporal logic of reactive systems. Springer Verlag (1991)
9. Reif, J., Sistla, A.P.: A multiprocess network logic with temporal and spatial modalities. *J. Comput. Syst. Sci.* **30**(1) (1985) 41–53
10. Browne, M.C., Clarke, E.M., Grumberg, O.: Reasoning about networks with many identical finite state processes. *Inf. Comput.* **81**(1) (1989) 13–31
11. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. *J. ACM* **39**(3) (1992) 675–735
12. Caires, L., Cardelli, L.: A spatial logic for concurrency (part II). In: Proceedings of the 13th International Conference on Concurrency Theory, Springer-Verlag (2002) 209–225
13. Caires, L., Cardelli, L.: A spatial logic for concurrency (part I). *Inf. Comput.* **186**(2) (2003) 194–235
14. Caires, L., Cardelli, L.: Behavioral and spatial observations in a logic for the π -calculus. In: FOSSACS'2004. Volume 2987 of Lecture Notes in Computer Science, Springer (2004) 72–87

15. Cardelli, L., Gordon, A.D.: Ambient logic. *Mathematical Structures in Computer Science* (2006) To appear.
16. Reynolds, J.: Separation logic: A logic for shared mutable data structures. In: *Logic in Computer Science*, IEEE Computer Society (2002) 55–74
17. Cardelli, L., Gardner, P., Ghelli, G.: Manipulating trees with hidden labels. In Gordon, A., ed.: *Foundations of Software Science and Computation Structures*. Volume 2620 of *Lect. Notes in Comp. Sci.*, Springer (2003) 216–232
18. Cardelli, L., Gardner, P., Ghelli, G.: A spatial logic for querying graphs. In Widmayer, P., Trigueiro Ruiz *et alii*, F., eds.: *Automata, Languages and Programming*. Volume 2380 of *Lect. Notes in Comp. Sci.*, Springer (2002) 597–610
19. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In Rozenberg, G., ed.: *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific (1997) 313–400
20. Dijkstra, E.W.: Hierarchical ordering of sequential processes. *Acta Inf.* **1** (1971) 115–138
21. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: *ICSE*. (1999) 411–420
22. Clarke, E., Grumberg, O., Peled, D.: *Model Checking*. The MIT Press (1999)
23. H. Vieira, L.C.: The spatial logic model checker user's manual. Technical Report TR-DI/FCT/UNL-03/2004, Faculty of Science and Technology New University of Lisbon (2004)
24. Berezin, S., Campos, S., Clarke, E.M.: Compositional reasoning in model checking. In: *COMPOS'97: Revised Lectures from the International Symposium on Compositionality: The Significant Difference*, London, UK, Springer-Verlag (1998) 81–102
25. Holzmann, G.J., Peled, D.: An improvement in formal verification. In Hogrefe, D., Leue, S., eds.: *FORTE*. Volume 6 of *IFIP Conference Proceedings.*, Chapman & Hall (1994) 197–211
26. Edelkamp, S., Leue, S., Lluch Lafuente, A.: Directed explicit-state model checking in the validation of communication protocols. *STTT* **5**(2-3) (2004) 247–267