Predicate Abstraction in Protocol Verification

Edgar Pek, Nikola Bogunović
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
E-mail: {edgar.pek, nikola.bogunovic}@fer.hr

***Abstract*— This paper presents how predicate abstraction can be applied to protocol verification. Predicate abstraction is a method for automatic construction of abstract state graph. Basic idea is to use n predicates $\phi_1, \ldots, \phi_n$ defined on concrete state space to generate abstract state graph. Model checking is a formal verification technique which has been successfully applied to protocol verification. But model checking can only be applied to finite state systems. Many interesting systems are infinite state or number of states is so large that verification becomes infeasible. Predicate abstraction can be applied in verification of infinite state systems (or large finite state systems). Abstract state graph created by predicate abstraction can be used for verification of safety properties using a model checker. We provide simple examples of protocol verification using predicate abstraction.**

I. Introduction

Correct functioning is implicit requirement for any system. Fulfilling that requirement is especially challenging for distributed software systems. Distributed software systems should appear as a coherent system in spite of being designed from many autonomous computers [1]. The crucial part of those systems is communication between components.

Communication can be made successful only if there are rules that should be followed by communicating entities. That rules are called protocols. Design of protocols that operate correctly is notoriously difficult. The main reason is that protocols must deal with asynchronous and concurrent computation in a heterogeneous environment.

Proper functioning of software system has usually been achieved using informal techniques such as simulation and testing. But those techniques can only be used to detect flaws. If we want to establish correctness, then formal methods based on mathematical logic must be employed.

In this paper we will be concerned with formal verification. There are two main approaches in formal verification: deductive and algorithmic verification. Deductive verification is a methodology in which correctness of the system is established using axioms and proof rules. Traditionally, proofs have been constructed entirely by hand. That process is very time consuming and error prone. Besides that, it requires considerable expertise in mathematics and logic. Over the time various tools have been developed which provided certain degree of automation. Those tools are known as theorem proving systems. In spite of automation, inherent characteristic of theorem proving is that usually requires considerable human intervention. However, theorem proving is a very powerful technique because we have all the methods of logic and mathematics at our disposal. We can verify any system we want, if provided with enough time and computational power.

Second methodology - algorithmic verification, is in large sense orthogonal to the previously described approach. The algorithmic verification methodology is best known as model checking [5]. Model checking is a technique for verification of finite state systems. The main idea is to perform exhaustive search of a state space to check whether specified correctness condition can be satisfied. The major advantage of the model checking is that verification can be carried out completely automatically. The main disadvantage is restriction to verification of finite state systems.

It is clear that model checking and theorem proving techniques complement each other. But, actual methodologies that combine these two approaches are still open research problems. In this article we will concentrate on one promising approach called predicate abstraction. Basic idea of our work has been to apply predicate abstraction to the problem of infiniteness that appears as main obstacle in protocol verification. We have concentrated on two aspects of the problem:

- infinite state space due to unbounded data types,
- real-time aspect.

Motivation for this work is related to our previous work. In [2] we have demonstrated verification of five mutual exclusion algorithms. All these algorithms were two process versions with finite data types. Abstractions used in that work had been obtained ad-hoc, without formal justification. It must be mentioned that all mutual exclusion algorithms are parameterized, which is the source of infiniteness, and cannot be tackled with classical predicate abstraction. So, we will not deal with it in this work. In our work related to verification of Bounded Retransmission protocol [3] we have obtained some under-approximations concerning both data types and timing aspects. Similarly in [4], we performed verification of the configuration in Logical Link Control and Adaptation Protocol from Bluetooth protocol stack. In all our work we have used only model checking [5] techniques based on symbolic state space representation by Binary Decision Diagrams [6].

Predicate abstraction has been introduced as a technique for reduction of an infinite state system to a finite state in the work of Graf and Saidi [7]. In that work, a finite state system is obtained as an over-approximation of an infinite state system. They have defined a technique

for generation of an abstract state graph and computation of an abstract reachable state space (invariants). Bounded Retransmission Protocol had been chosen as a case study, but only basic facts about actual verification were provided.

Another verification technique based on the predicate abstraction had been proposed by Colón and Uribe [8]. In their work predicate abstraction is done on transitions rather then on state space. Although potentially more efficient than techniques which abstract state space that technique usually yields coarser abstraction. Their work has been implemented in a tool STeP (Stanford Temporal Prover), but version of the tool that supports predicate abstraction is not available.

Predicate abstraction in terms of protocol verification has been studied by David Dill's group at Stanford. They have verified various protocols using tool $Mur\phi^{--}$. Some results are reported in [9]. Most of the work is described in context of various approaches to predicate abstraction, and examples are in $Mur\phi^{--}$ input language. That tool is also not available, and there is no documentation about the language. So, we have not been able to fully understand the provided examples.

We have found the work of Saidi and Shankar [10] and review by Shankar [11] to be most useful for our purpose. Also, the idea of predicate abstraction as syntactic transformation (as in [12]) is used in our work.

The rest of paper is organised as follows. In Section II predicate abstraction will be described based on the [10] and [11]. Section III presents a verification of the Bakery mutual exclusion algorithm for two processes. In Section IV the Fischer mutual exclusion algorithm is used as a case study for real time verification by predicate abstraction. Additional insights about real-time verification based on predicate abstraction came from work by Möller, Rueß and Sorea [13]. Finally, in Section V we will give some concluding remarks.

II. Predicate Abstraction

Intuitively, predicate abstraction provides a mapping from a concrete to an abstract system. Concrete system is usually an infinite state or has extremely large number of states. Abstract system is a finite state, where states correspond to truth assignments to a set of predicates (defined on the concrete system). Now, we will provide a formal description of the predicate abstraction.

Predicate abstraction is a methodology which is based on the abstract interpretation [14]. Abstract interpretation is the framework for defining abstractions. It is based on the Galois connection.

Definition 1 (Galois connection). *Galois connection is a pair (α, γ) that defines a mapping between a concrete domain lattice $\mathcal{P}(\mathcal{Q})$ and an abstract domain lattice $\mathcal{P}(\mathcal{Q}^A)$. Where α and γ represent two monotonic functions such that:*

$$\forall (P_1, P_2) \in \mathcal{P}(\mathcal{Q}) \times \mathcal{P}(\mathcal{Q}^A) \mid \alpha(P_1) \subseteq P_2 \Leftrightarrow P_1 \subseteq \gamma(P_2).$$

The domain of the abstraction function α is poset (i.e. a partially ordered set) $\mathcal{P}(\mathcal{Q}) \equiv (S, \Rightarrow)$, where S represent sets of concrete states, ordered by $\Rightarrow$ (implication). Range of α are boolean formulas built from boolean variables $B_1, B_2, \ldots, B_k$. Let X and Y represent a domain of abstraction (α) and concretization (γ) function respectively. Functions γ and α can be implicitly defined as:

$$\gamma(Y) = \bigvee \{X \mid \alpha(X) \Rightarrow Y\}, \tag{1}$$

$$\alpha(X) = \bigwedge \{Y \mid X \Rightarrow \gamma(Y)\}. \tag{2}$$

In abstract space each boolean variable B_i represents all concrete states that satisfy predicate ϕ_i. So, the concretization function can be obtained simply by replacing each abstract variable B_i with a corresponding predicate ϕ_i, and each abstract state variable with a corresponding concrete state variable. Thus, simpler definition of the function γ is:

$$\gamma(Y) = Y[\phi_i(s)/B_i(abs_s)]. \tag{3}$$

The computation of the abstraction function is much more intricate. As it was mentioned in the Section I there are various approaches to the problem. We will present a method described in [10] and [11].

Based on the definition of abstraction function (2) we can conclude the following. For any predicate P over the concrete variables, the abstraction $\alpha(P)$ of P can be computed as the conjunction of all boolean expressions satisfying the condition:

$$P \Rightarrow \gamma(b). \tag{4}$$

There are 2^{2^k} distinct boolean functions in k variables (computation of (4) for all functions becomes very expensive). This set of functions is known as a set of *test points*. An abstraction is *precise* with respect to the considered abstract lattice if the set of test points is the entire set of boolean expressions that form an abstract lattice. Smaller set of test points can be used, but equation (4) must still be valid. All those smaller sets represent over-approximations (coarser approximations). The first example of predicate abstraction [7] used the lattice of monomials[1] as the abstract lattice.

Saidi and Shankar [10] provided a method for computation of the abstraction function without the loss of precision which is more efficient (i.e. it requires fewer number of tests). To accomplish the goal they have chosen appropriate sub-lattice and test points (see Theorem 1).

Theorem 1 (Sub-lattice and test points). *Let $B = \{B_1, \ldots, B_k\}$ be a set of boolean variables, and let $\mathcal{B}_A$ be the boolean algebra defined by the structure $\langle B, \wedge, \vee, \neg, true, false \rangle$. Let $\mathcal{D}_B$ be the subset of $\mathcal{B}_A$ containing only literals and disjunctions of literals. To compute the most precise image by α of any set of concrete states P (given as a predicate), it is sufficient to consider as a set of test points, the set $\mathcal{D}_B$ instead of the whole set $\mathcal{B}_A$ of boolean expressions. That is, testing*

$$P \Rightarrow \gamma(b) \tag{5}$$

[1]Monomial is a conjunction of b_i's where b_i can be either B_i or $\neg B_i$. Each b_i can appear exactly once.

for all boolean expressions in $\mathcal{B}_A$ is equivalent to test this implication only for b in $\mathcal{D}_B$. So, 2^{2^k} tests can be reduced to at most $3^k - 1$ tests.

Proof: Proof is based on the fact that each boolean expression can be written in a conjunctive normal form (CNF) as $d_1 \wedge \ldots \wedge d_j$, where d_i represent disjunction of literals. So, proof of implication (5) for each element b can be decomposed in proofs for d_i's. Thus, by testing only elements in $\mathcal{D}_B$ we can cover all boolean truth functions.

On the grounds of the above theorem the computation of precise abstraction is reduced from 2^{2^k} to $3^k - 1$ (recall that k is the number of boolean variables representing predicates). Moreover, actual number of tests can be reduced because some of the tests for the elements $d_i \in \mathcal{D}_B$ become redundant (e.g. because of subsumption). In the following example we will illustrate the above theory.

a) Example: Suppose that we want to abstract a formula $x = y$, which for example appears in a transition relation of a protocol. Variables x and y are integers. First, we must define predicates on the concrete space:

$$\phi_1 \equiv x > 0$$
$$\phi_2 \equiv y > 0$$

Furthermore, let a and b represent boolean variables in the abstract lattice.

According to (3) concretization function is defined with $\gamma(a) = x > 0$ and $\gamma(b) = y > 0$.

Based on the theorem 1 we create disjunctions d_i and for each d_i test if $x = y \Rightarrow \gamma(d_i)$ is provable.

TABLE I

DISJUNCTS d_i AS TEST POINTS

d_i	$\vdash^? x = y \Rightarrow \gamma(d_i)$
a	$\not\vdash x = y \Rightarrow x > 0$
$\neg a$	$\not\vdash x = y \Rightarrow x \not> 0$
b	$\not\vdash x = y \Rightarrow y > 0$
$\neg b$	$\not\vdash x = y \Rightarrow y \not> 0$
$a \vee b$	$\not\vdash x = y \Rightarrow x > 0 \vee y > 0$
$a \vee \neg b$	$\vdash x = y \Rightarrow x > 0 \vee y \not> 0$
$\neg a \vee b$	$\vdash x = y \Rightarrow x \not> 0 \vee y > 0$
$\neg a \vee \neg b$	$\not\vdash x = y \Rightarrow x \not> 0 \vee y \not> 0$

Based on the results shown in the table I we can create the abstraction of the atomic formula $x = y$:

$$\alpha(x = y) = (a \vee \neg b) \wedge (\neg a \wedge b). \tag{6}$$

If we enumerate disjunctions in order of increasing length (as it was done in table I) then we can reduce the number of tests. If the test fails on a disjunction d_i but succeeds on the disjunction $d_i \vee q$ then we can skip tests for:

1) the disjunction $d_i \vee \neg q$, because if this test succeeded the d_i would have also succeeded, and so this test can be eliminated,
2) any disjunction that extends $d_i \vee q$, since these are weaker approximations, and therefore subsumed by $d_i \vee q$.

Also, it is not necessary to consider any literals q where $\gamma(q)$ and p share no variables since q is irrelevant and cannot contribute to the success of the test.

For instance, the last test $\neg a \vee \neg b$ in the table I could be eliminated because $a \vee \neg b$ succeeded while $\neg b$ didn't.

In this section we have shown theory behind predicate abstraction. In the following two sections we will apply the theory on the two examples of mutual exclusion.

III. BAKERY MUTUAL EXCLUSION EXAMPLE

In this section we will demonstrate predicate abstraction of the two process Bakery mutual exclusion protocol. This example will show how predicate abstraction can be used to tackle problem of infinite state space due to unbounded data types. First, we will provide description of the protocol, then results of predicate abstraction. The results of predicate abstraction have been verified by a symbolic model checker called NuSMV [15].

A. Two process Bakery mutual exclusion protocol

Protocol 1 Bakery mutual exclusion protocol

var y_1, y_2: integer;
initially $y_1 = y_2 = 0$
P_1 :
while true **do**
  $l0$: <noncritical section>;
  $l1$: $y_1 := y_2 + 1$;
  $l2$: **await** $y_2 = 0 \ \vee \ y_1 < y_2$;
  $l3$: <critical section>;
  $l4$: $y_1 := 0$;
end while
P_2 :
while true **do**
  $m0$: <noncritical section>;
  $m1$: $y_2 := y_1 + 1$;
  $m2$: **await** $y_1 = 0 \ \vee \ y_2 \leq y_1$;
  $m3$: <critical section>;
  $m4$: $y_2 := 0$;
end while

From the above description it can be noted that protocol cannot be verified using finite state techniques. The main reason is that the variables y_1 and y_2 are unbounded. Note that, even if actual implementation is considered, state space is very large for only two variables.

B. Predicate abstraction of Bakery protocol

As it has been shown in Section II, the first step in predicate abstraction is definition of the predicates. In this example we check only the basic property of mutual exclusion, expressed as: both processes cannot be in the critical section at the same time. Temporal logic formula (expressed in Computation Tree Logic - see e.g. [5]) for this property can be written as:

$$AG \ \neg(P1.pc = l3 \ \wedge \ P2.pc = m3). \tag{7}$$

Usually, a temporal logic formula helps us to define a suitable set of predicates. In this simple example, the set of predicates is easily determined by looking at the

condition which guards critical section ($l2$ and $m2$). Based on this we propose the following set of predicates:

$$\phi_1 \equiv y_1 = 0$$
$$\phi_2 \equiv y_2 = 0$$
$$\phi_3 \equiv y_1 < y_2$$

Boolean variables that represent the abstract lattice are: B_1, B_2, B_3. Concretization function γ is defined with $\gamma(B_1) = (y_1 = 0)$, $\gamma(B_2) = (y_2 = 0)$ and $\gamma(B_3) = (y_1 < y_2)$. Now the abstraction function must be defined. Let us consider transition defined as $y_1 := y_2 + 1$, for which we can compute the abstraction function as described in the example provided in Section II.

TABLE II

ABSTRACTION OF $y_1 := y_2 + 1$

d_i	$\vdash^? y_1 := y_2 + 1 \Rightarrow \gamma(d_i)$
B_1	$\not\vdash y_1 := y_2 + 1 \Rightarrow y_1 = 0$
$\neg B_1$	$\vdash y_1 := y_2 + 1 \Rightarrow y_1 \neq 0$
B_3	$\not\vdash y_1 := y_2 + 1 \Rightarrow y_2 = 0$
$\neg B_3$	$\vdash y_1 := y_2 + 1 \Rightarrow y_2 \neq 0$
$B_1 \vee B_3$	$\not\vdash y_1 := y_2 + 1 \Rightarrow (y_1 = 0 \vee y_2 = 0)$
$B_1 \vee \neg B_3$	$subsumed$
$\neg B_1 \vee B_3$	$subsumed$
$\neg B_1 \vee \neg B_3$	$subsumed$

Based on the results in the table II, the abstraction of transition $y_1 := y_2 + 1$ is:

$$\alpha(y_1 := y_2 + 1) = \neg B_1 \wedge \neg B_3. \quad (8)$$

Equation (8) is equivalent to following assignments: $B_1 := 0$ and $B_3 := 0$. Note, that we don't need to consider variable B_2 since it is not affected with transition relation. Similarly, we have abstracted all other transitions. The abstracted version of Bakery mutual exclusion protocol is shown as Protocol 2.

Protocol 2 Abstraction of Bakery mutual exclusion protocol

var B_1, B_2, B_3: boolean;
initially $B_1 = B_2 = 1, B_3 = 0$
P_1 :
while true **do**
  $l0$: <noncritical section>;
  $l1$: $B_1 := 0,\ B_3 := 0$;
  $l2$: **await** $B_2 \vee B_3$;
  $l3$: <critical section>;
  $l4$: $B_1 := 1,\ B_3 := \neg B_2$;
end while
P_2 :
while true **do**
  $m0$: <noncritical section>;
  $m1$: $B_2 := 0,\ B_3 := 1$;
  $m2$: **await** $B_1 \vee \neg B_3$;
  $m3$: <critical section>;
  $m4$: $B_2 := 1,\ B_3 := 0$;
end while

We have translated the obtained version of the Bakery mutual exclusion protocol into the NuSMV input language. Using NuSMV engine for symbolic model checking with BDD's, mutual exclusion (7) property has been successfully verified. NuSMV files can be found on the web page: http://fmg.zemris.fer.hr/PredicateAbstraction/Bakery/.

IV. FISCHER MUTUAL EXCLUSION EXAMPLE

This section will provide predicate abstraction of the Fischer's real-time mutual exclusion protocol. In this example predicate abstraction will be used to abstract a real-time system to a finite state system, which is amenable to model checking. The source of infiniteness in this example is caused by real-time aspects of the protocol. As in the previous section, we give a protocol description first and then results of the predicate abstraction.

A. Two process Fischer mutual exclusion protocol

Fischer's mutual exclusion protocol (shown as Protocol 3) assumes uniform positive bounds on time each process can wait before executing its next statement: an enabled transition must wait at least L and at most U before being taken. If $2L > U$ the protocol guarantees that both processes are never in their critical sections simultaneously.

Protocol 3 Fischer's mutual exclusion protocol

var $x : \{0, 1, 2\}$;
initially $x = 0$
P_1 :
while true **do**
  $l0$: **await** $x = 0$;
  $l1$: $x := 1$;
  $l2$: **skip**;
  $l3$: **await** $x = 1$;
  $l4$: <critical section>;
  $l5$: $x := 0$
end while
P_2 :
while true **do**
  $m0$: **await** $x = 0$;
  $m1$: $x := 2$;
  $m2$: **skip**;
  $m3$: **await** $x = 2$;
  $m4$: <critical section>;
  $m5$: $x := 0$
end while

Note that in Fischer's real time mutual exclusion protocol we don't have to deal with unboundness of data types (variable x has a finite domain).

B. Predicate abstraction of Fischer's protocol

Besides theory provided in Section II, we will use insights from the work of Möller, Rueß and Sorea [13] as well as from Colon and Uribe [8].

To model real-time systems it is necessary to augment classical state transition systems with a finite set of real valued clocks. The clocks proceed at a uniform rate and constrain the times at which transition may occur. Those

transition systems are called *timed automata*. A formal definition of timed automata can be found in [13].

As it has been mentioned in the previous section, one of the main problems with predicate abstraction is to define appropriate set of predicates. In the case of timed automata it was shown in [13] that the set of abstraction predicates expressive enough to distinguish between any two clock regions determines a strongly preserving abstraction. The set is called basis.

The main technical problem in the definition of the abstraction is to guarantee fairness in the abstract model; that is, to prevent delay steps to be abstracted into self-loops on the abstract system. In the work of [13] that problem is addressed by introducing restricted delay steps, while in [8] there is a notion of time progress condition (in the context of clocked transition systems). For our purpose both approaches are equivalent. The only important thing is to restrict progress of time.

First, we define the basis for the Fischer's protocol.

$$\begin{aligned}\Phi :=&\{c_1 = 0, c_2 = 0, c_1 \leq L, c_2 \leq L,\\ &c_1 \geq L, c_2 \geq L, c_1 \leq U, c_2 \leq U\}\end{aligned} \tag{9}$$

Where, c_1 and c_2 represent first resp. second process clock variables.

We don't have to use all predicates from the basis, but rather we can incrementally add predicates until abstraction is fine enough. That process is known as a stepwise refinement.

In our example, two predicates from the basis set Φ (9) have been initially chosen:

$$\phi_1 \equiv c_1 \geq L$$
$$\phi_2 \equiv c_2 \geq L$$

Based on those two predicates we get the abstract version of the Fischer's mutual exclusion protocol. The procedure for obtaining the concretization and the abstraction function is same as in Section III.

The abstraction of the Fischer's mutual exclusion protocol (Protocol 4) must be augmented with a tick transition. The abstraction of the tick transition just changes variables B_1 and B_2. Variables can be non-deterministically changed only when B_1 and B_2 are $false$.

As in previous section, the protocol has been described and verified using the NuSMV system. NuSMV input files for that example are available from: http://fmg.zemris.fer.hr/PredicateAbstraction/Fischer/.

Protocol 4 Abstraction of Fischer's mutual exclusion protocol

var $x : \{0, 1, 2\};\ B1, B2$: boolean;
initially $x = 0$, $B1 = B2 = false$
P_1 :
while true **do**
  $l0$: **await** $(B_1 \wedge x = 0)$;
  $l1$: $x := 1$;
  $l2$: **await** B_1;
  $l3$: **await** B_1;
  $l4$: **await** $x = 1$;
  $l5$: <critical section>;
  $l6$: $x := 0$
end while
P_2 :
while true **do**
  $m0$: **await** $(B_2 \wedge x = 0)$;
  $m1$: $x := 2$;
  $m2$: **await** B_2;
  $m3$: **await** B_2;
  $m4$: **await** $x = 2$;
  $m5$: <critical section>;
  $m6$: $x := 0$
end while

V. Conclusion

In this work we have presented how predicate abstraction can be applied to protocol verification. Predicate abstraction is a methodology that combines two orthogonal verification approaches: theorem proving and model checking. It relates powerful decision procedures from theorem proving and automatic property checking on a finite model from model checking.

The first problem when applying predicate abstraction is defining appropriate set of predicates. Currently, there is no proper formal procedure that would provide a suitable set of predicates for a given protocol description. However, useful predicates can be obtained from a model described with guarded commands. In that case, guards are usually chosen as predicates. Besides that, property being verified provides us with predicates that can be useful. The process of finding good predicates still relies on experience and domain knowledge.

The second issue in predicate abstraction is the creation of abstraction. Abstraction can be created on various levels. The first work [7] on predicate abstraction created abstraction as a state graph. The approach described in [12] considered abstraction on syntactic level. Our approach is similar to the second approach, because other techniques (symbolic model checking, partial-order reduction) can be applied to the obtained abstraction. Besides that, we must choose the appropriate abstract lattice which must be expressive enough, but also should not generate to much validity checks. Validity checks, done by theorem prover, are the most expensive part in a computation of abstraction.

From the aspects of properties that can be verified we have been concerned only with properties expressible as temporal logic formulas without existential quantification over paths. These properties mostly fall in the class of safety properties.

Two examples have demonstrated how predicate abstraction can be used to reduce an infinite to a finite system. In the first example it has been shown how the Bakery mutual exclusion protocol can be abstracted to a finite state protocol. The main source of infiniteness in this example has been unbounded data types. The results have shown that predicate abstraction is just the right approach for that class of systems. The second example has demonstrated more subtle source of infiniteness, related to real-

time aspects of the Fischer's mutual exclusion protocol. Predicate abstraction can be successfully applied to this kind of systems, but some rules must be followed. First, one must choose a set of predicates which is expressive enough to distinguish between two clock regions. Second, progress of time must be restricted by some conditions to ensure fairness.

The main problem that we will try to tackle in our further work is infiniteness due to parameterization. Most protocols are parameterized, so this is very important issue. One of the ideas is to use quantified predicates in the process of predicate abstraction. The main problem is how to relate different instances of the same protocol.

REFERENCES

[1] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ: Prentice Hall, 2002.

[2] N. Bogunović and E. Pek, "Verification of mutual exclusion algorithms with SMV system," in *Proceedings of IEEE Region 8 Eurocon 2003: Computer as a Tool*, B. Zajc and M. Tkalčić, Eds., vol. II. IEEE, September 2003., pp. 12–25.

[3] E. Pek and N. Bogunović, "Formal verification of communication protocols in distributed systems," in *MIPRO 2003, Proceedings of the Joint Conferences Computers in technical systems and Intelligent systems*, B. Leo and R. Slobodan, Eds., May 2003., pp. 44–49.

[4] ——, "Formal verification of logical link control and adaptation protocol," in *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference, MELECON 200*, M. Maja, P. Branimir, T. Željko, and B. Željko, Eds., May 2004., pp. 583–586.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: The MIT Press, 1999.

[6] K.L. McMillan, "Symbolic Model Checking: An Approach to the State Explosion Problem," Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1992, CMU-CS-92-131.

[7] S. Graf and H. Saïdi, "Construction of abstract state graphs with pvs," in *CAV '97: Proceedings of the 9th International Conference on Computer Aided Verification*. Springer-Verlag, 1997, pp. 72–83.

[8] M. Colón and T. E. Uribe, "Generating finite-state abstractions of reactive systems using decision procedures," in *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*. Springer-Verlag, 1998, pp. 293–304.

[9] S. Das, D. L. Dill, and S. Park, "Experience with predicate abstraction," in *11th International Conference on Computer-Aided Verification*. Springer-Verlag, July 1999.

[10] H. Saïdi and N. Shankar, "Abstract and model check while you prove," in *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*. Springer-Verlag, 1999, pp. 443–454.

[11] N. Shankar, "Automated verification using deduction, exploration, and abstraction," pp. 333–351, 2003.

[12] S. Bensalem, Y. Lakhnech, and S. Owre, "Computing abstractions of infinite state systems compositionally and automatically," in *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*. Springer-Verlag, 1998, pp. 319–331.

[13] M. O. Möller, H. Rueß, and M. Sorea, "Predicate abstraction for dense real-time systems," *Electronic Notes in Theoretical Computer Science*, vol. 65, no. 6, 2002, full version available as Technical Report BRICS-RS-01-44, Department of Computer Science, University of Aarhus, Denmark.

[14] P. Cousot and R. Cousot, "Abstract intrepretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*, Los Angeles, CA, Jan. 1977, pp. 238–252.

[15] R. Cavada, A. Cimatti, E. Olivetti, M. Pistore, and M. Roveri, *NuSMV 2.1 User Manual*, 2002. [Online]. Available: http://nusmv.irst.itc.it/NuSMV/userman/v21/nusmv.pdf

[16] S. Das, "Predicate Abstraction," Ph.D. dissertation, Department of Computer Science, Stanford University, Stanford, CA, December 2003.