# Formal Verification Coverage: Computing the Coverage Gap between Temporal Specifications

**Sayantan Das** **Prasenjit Basu** **Ansuman Banerjee**
**Pallab Dasgupta** **P.P. Chakrabarti**
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur, India.

**Chunduri Rama Mohan**
Strategic DA Planning, Client Platform Division
Intel Corporation, Folsom, USA.

**Limor Fix** **Roy Armoni**
Logic and Validation Technology
Intel Corporation, Haifa, Israel.

## Abstract

*Existing methods for formal verification coverage compare a given specification with a given implementation, and evaluate the coverage gap in terms of quantitative metrics. In this paper, we consider a new problem, namely to compare two formal temporal specifications and to find a set of additional temporal properties that close the coverage gap between the two specifications. In this paper we present: (1) the problem definition and motivation, (2) a methodology for computing the coverage gap between specifications, and (3) a methodology for representing the coverage gap as a collection of temporal properties that preserve the syntactic structure of the target specification.*

## INTRODUCTION

In recent times, formal property verification (FPV) is finding increased acceptance within the pre-silicon design validation flow. Design intent validation (alternatively, the task of verifying whether the RTL meets the designer's architectural intent), continues to pose new challenges to the design flow today. Traditional approaches to formal verification attempt to validate the given RTL implementation against a formal specification using techniques such as model checking [4].

While FPV guarantees *exhaustive* validation of a formal property on a given implementation, the task of determining the completeness of the specification itself is left open. The aim of FPV coverage is to close this gap. Existing methods for FPV coverage [1, 3, 5, 6] compare a given specification with a given implementation and deduce: (1) the parts of the implementation that are not covered by the specification and (2) the properties in the specification that are vacuously satisfied in the implementation. The first measure indicates incompleteness of the specification [5], while the second measure indicates incompleteness of the implementation [1] (in the sense that some part of the behavior modeled by the formal specification is vacuous in the implementation).

In this paper, we study a new FPV coverage problem where we compare two formal specifications and determine the coverage of one by the other. We shall refer to one specification as the *target specification*, $\mathcal{T}$, and the other as the *achieved specification*, $\mathcal{A}$. Our aim will be to determine whether the *achieved specification* covers the *target specification*. If the answer is negative, then we will further aim to determine the coverage gap or difference, $\mathcal{T} - \mathcal{A}$, between the specifications, and represent it in a way that is both legible and syntactically comparable with $\mathcal{T}$. The problem is non-trivial, since both $\mathcal{T}$ and $\mathcal{A}$ may contain several temporal properties.

We believe that the core problem of determining the coverage of one specification by another will play an important role in formal verification. Foreseeable applications include:

- Coverage of system-level properties by RTL properties of component modules. The system-level properties define the *target specification*, $\mathcal{T}$, and the RTL properties collectively define the *achieved specification*, $\mathcal{A}$.
- Specification matching. In order to determine whether a target specification can be implemented by an existing IP core, we need to determine whether the properties guaranteed by the IP core cover the target specification.

One of the main issues in the given coverage problem is to find an appropriate representation for the coverage gap. Typically, individual counter-example traces that satisfy the achieved specification but refute the target specification, expose only a small fraction of the coverage gap. On the other hand, we aim to express the coverage gap in terms of additional temporal properties that succinctly encapsulate all the counter-example traces. We further aim to present these additional temporal properties in a style that is syntactically similar to the target specification, so that the designer can *visually* compare them with the target specification. Section presents examples to explain this intent.

A restricted version of the specification matching problem was introduced by us as a short interactive presentation in a recent conference [2]. Since then we have made significant progress in developing the methodology for solving the coverage problem. This paper presents a new and complete methodology for the coverage problem, where the specifications are given in *Linear Temporal Logic* [7] (LTL).

0-7803-8702-3/04/$20.00 ©2004 IEEE.

## THE SPECIFICATION COVERAGE PROBLEM

The inputs to the specification coverage problem are:

- The *target specification* $\mathcal{T}$ as a set of LTL properties over a set, $\mathcal{AP}_T$, of Boolean signals, and
- The *achieved specification* $\mathcal{A}$ as another set of LTL properties over a set, $\mathcal{AP}_A$, of Boolean signals.

We shall also use $\mathcal{T}$ to denote the conjunction of the properties in the target specification, and $\mathcal{A}$ to denote the conjunction of the properties in the achieved specification.

**Assumption 1** *Throughout this paper we assume that* $\mathcal{AP}_T \subseteq \mathcal{AP}_A$.

Typically this is not a restrictive assumption within the design hierarchy, since it is generally considered a good practice for designers at a lower level of the design hierarchy to inherit the interface signal names from the previous level of hierarchy.

Given a specification, we define a *state* as a valuation of the signals used in the specification. A *run* is an infinite sequence of states.

**Definition 1 [Coverage Definition: ]**
*The achieved specification covers the target specification iff there exists no run that refutes one or more properties of the target specification but does not refute any property of the achieved specification.* □

Our coverage problem is as follows:

- To determine whether the achieved specification covers the target specification, and
- If the answer to the previous question is *no*, then to determine a set of additional temporal properties that represent the coverage gap (that is, these properties together with the achieved specification succeed in covering the target specification).

The following theorem shows us a way to answer the first question[1].

**Theorem 1** *The achieved specification,* $\mathcal{A}$*, covers the target specification* $\mathcal{T}$*, iff the temporal property* $\mathcal{A} \Rightarrow \mathcal{T}$ *is valid.* □

The theorem shows that the primary coverage question can be answered by testing the validity of $\mathcal{A} \Rightarrow \mathcal{T}$. Most model checking tools for LTL already have the capability of performing this validity check.

**Example 1** Let us consider the design of an arbiter that arbitrates between two request lines $r_1$ and $r_2$ from two master devices. Let the corresponding grant lines to the master devices be $g_1$ and $g_2$. The arbiter also receives an input $z$ from a slave device, that remains high as long as the slave device is *ready*.

[1] Proofs of all theorems are available in the detailed version of this paper. Contact: pallab@cse.iitkgp.ernet.in

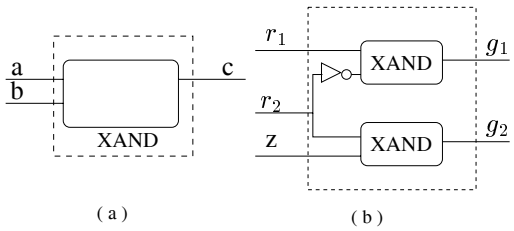


**Figure 1. A sample arbiter**

The arbiter specification requires us to treat $r_2$ as a high-priority request. Whenever $r_2$ is asserted and the slave is ready (that is, $z$ is high), the arbiter must give the grant, $g_2$ in the next cycle, and continue to assert $g_2$ as long as $r_2$ remains asserted. When $r_2$ is not high, the arbiter parks the grant on $g_1$ regardless of whether $r_1$ is asserted. We are further given, that the request $r_2$ is fair in the sense that it is de-asserted infinitely often (enabling $g_1$ to be asserted infinitely often). The target specification (in LTL) is:

$$\begin{array}{ll} T_1\text{:} & G\ F(\ \neg r_2\ ) \\ T_2\text{:} & G(\ (\ r_2\ \wedge\ z\ )\ \Rightarrow\ X(\ g_2\ U\ \neg r_2\ )\ ) \\ T_3\text{:} & G(\ (\ \neg r_2\ )\ \Rightarrow\ X\ g_1\ ) \end{array}$$

Let us now consider an implementation of the arbiter using an existing component called XAND, as shown in Fig 1. The specification of the module XAND is as follows:

$$A_1'\text{:}\quad G(\ (\ a \wedge\ b\ )\ \Rightarrow X\ c\ )\ )$$

Substituting the signal names of the instances of XAND in Fig 1(b) with $r_1$, $r_2$, $g_1$, $g_2$ and $z$, and adding the fairness property on $r_2$, we have the achieved specification as:

$$\begin{array}{ll} A_1\text{:} & G\ F(\ \neg r_2\ ) \\ A_2\text{:} & G(\ (\ r_1\ \wedge\ \neg r_2\ )\ \Rightarrow\ X\ g_1\ ) \\ A_3\text{:} & G(\ (\ r_2\ \wedge\ z\ )\ \Rightarrow\ X\ g_2\ ) \end{array}$$

The first property is the same fairness constraint as in the target specification. The second property says if $r_1$ is asserted and $r_2$ is deasserted then $g_1$ is asserted in the next cycle. The third property states that whenever $r_2$ and $z$ are asserted together, then $g_2$ is asserted in the next cycle.

Our primary coverage problem is to determine whether $(A_1 \wedge A_2 \wedge A_3) \Rightarrow (T_1 \wedge T_2 \wedge T_3)$ is valid. In this case the answer is negative, since neither $T_2$ nor $T_3$ is covered by the properties in the achieved specification. For example whenever we have a scenario where both $r_1$ and $r_2$ are low, the target specification requires $g_1$ to be asserted, but the achieved specification does not have this requirement. This shows that $T_3$ is not covered. Also, consider those scenarios where $r_2$ and $z$ are asserted together, but $z$ deasserts before $r_2$ (that is, the slave becomes unavailable before the transfer completes). In such scenarios, property $T_2$ requires $g_2$ to remain

high as long as $r_2$ remains high, but the achieved specification does not guarantee this. □

## COMPUTING THE COVERAGE GAP

We now address the more complex problem of computing and representing the coverage gap. One way to demonstrate that a coverage gap exists is to produce a counter-example run, that is, a run that satisfies the achieved specification but refutes the target specification. However, this only reflects a fraction of the coverage gap. On the other hand, our aim is to find the set of missing temporal properties in the achieved specification, which when included in the achieved specification enables the coverage of the target specification.

**Example 2** Let us consider the coverage of the property, $T_3$ of Example 1 by the achieved specification. We have already established that $T_3$ is not covered. However this information does not accurately point out the coverage gap between $T_3$ and the achieved specification. Specifically the coverage gap lies only for those scenarios where $r_1$ and $r_2$ are simultaneously low at some point of time. In other words, the coverage gap can be accurately represented by the following property that considers exactly the above scenarios:

$$U_1: \quad G(\ (\ \neg r_1\ \wedge\ \neg r_2\ )\ \Rightarrow\ X\ g_1\ )$$

We have $A_2 \wedge U_1 \Rightarrow T_3$, and therefore $U_1$ closes the coverage gap between $A_2$ and $T_3$. In general, our aim will be to determine the *weakest* set of temporal properties that close the coverage gap between the achieved specification and the target specification. □

**Definition 2 [Strong and weak properties: ]**
*A property $\mathcal{F}_1$ is stronger than a property $\mathcal{F}_2$ iff $\mathcal{F}_1 \Rightarrow \mathcal{F}_2$ and $\mathcal{F}_2 \not\Rightarrow \mathcal{F}_1$. We also say that $\mathcal{F}_2$ is weaker than $\mathcal{F}_1$.* □

**Definition 3 [Coverage Hole in Achieved Specification: ]**
*A coverage hole in the achieved specification is a property $A_H$ over $\mathcal{AP}_A$, such that $(\mathcal{A} \wedge A_H) \Rightarrow \mathcal{T}$, and there exists no property, $A'_H$, over $\mathcal{AP}_A$ such that $A'_H$ is weaker than $A_H$ and $(\mathcal{A} \wedge A'_H) \Rightarrow \mathcal{T}$. In other words, we find the weakest property that suffices to close the coverage hole. Adding the weakest property strengthens the achieved specification in a minimal way.* □

Since $\mathcal{AP}_T \subseteq \mathcal{AP}_A$, each property of the target specification is a valid property over $\mathcal{AP}_A$. The following theorem characterizes the coverage hole.

**Theorem 2** *The coverage hole in the achieved specification is unique and is given by $\mathcal{T} \vee \neg\mathcal{A}$.* □

The following example demonstrates the notion of a coverage hole in our formulation.

**Example 3** Let us again consider the arbiter of Example 1. We have seen that the coverage gap lies in $T_2$ and $T_3$. By Theorem 2 we have the coverage hole in the achieved specification as:

$$A_H: \quad (\ (T_2 \wedge T_3) \vee \neg(A_1 \wedge A_2 \wedge A_3)\ )$$

We can also write the same coverage hole as the conjunction of the following two properties:

$$A_H{}': \quad (\ T_2 \vee \neg(A_1 \wedge A_2 \wedge A_3)\ )$$
$$A_H{}'': \quad (\ T_3 \vee \neg(A_1 \wedge A_2 \wedge A_3)\ )$$ □

The coverage hole, $\mathcal{T} \vee \neg\mathcal{A}$, may contain signals belonging to $\mathcal{AP}_A - \mathcal{AP}_T$. In many cases, the designer may want to evaluate the part of the target specification that is not covered by the achieved specification. We therefore also consider the problem of computing the *uncovered target specification* as defined below.

**Definition 4 [Uncovered Target Specification: ]**
*An uncovered target specification is a property $T_H$ over $\mathcal{AP}_T$, such that $(\mathcal{A} \wedge T_H) \Rightarrow \mathcal{T}$, and there exists no property $T'_H$ over $\mathcal{AP}_T$ such that $T'_H$ is weaker than $T_H$ and $(\mathcal{A} \wedge T'_H) \Rightarrow \mathcal{T}$. In other words, we find the weakest property over $\mathcal{AP}_T$ that suffices to close the coverage hole.* □

## REPRESENTING THE COVERAGE HOLE

Our aim is to present the coverage hole and the uncovered target specification before the designer in a form that is syntactically close to the target specification and is thereby amenable to visual comparison with the target specification. The expressibility of the logic used for specification does not always permit a succinct representation of the coverage hole. In such cases, we prefer to present the coverage hole as a succinct set of properties that closes the coverage gap, but may be marginally stronger than the coverage gap.

**Example 4** We consider the coverage of $T_3$ by $A_2$ in the specifications given in Example 1. By Theorem 2, the coverage gap between $T_3$ and $A_2$ is given by the property: $\varphi = T_3 \vee \neg A_2$, which does not convey any meaningful information to the designer. On the other hand, consider the property $U_1$ of Example 2:

$$U_1: \quad G(\ (\ \neg r_1\ \wedge\ \neg r_2\ )\ \Rightarrow\ X\ g_1\ )$$

$U_1$ is stronger than $\varphi$, but is able to represent the coverage gap more effectively than $\varphi$, because the designer can visually compare $U_1$ with $T_3$ to examine the coverage gap.

It is also important to be able to preserve structural similarity with the target specification when we present the coverage hole. For example, the property $U_1$ can also be written as:

$$G(\ r_1\ \vee\ r_2\ \vee X\ g_1\ )$$

or as:

$$G(\ (\ \neg X\ g_1\ \wedge\ \neg r_1\ )\ \Rightarrow\ r_2\ )$$

These representations are logically equivalent to $U_1$, but are not visually similar to $T_3$. We feel that preserving structural similarity is a very important issue in representing the gaps between formal specifications. □

Our specification coverage methodology is based on two key algorithms. The first algorithm enables the computation of the bounded terms in the coverage gap and then *pushes* these terms into the syntactic structure of the target properties to obtain the uncovered part. The second algorithm takes target properties having unbounded temporal operators (such as $G$, $F$ and $U$) and systematically weakens them into structure preserving decompositions. It then checks the components that remain to be covered. The following example illustrates this approach.

**Example 5** Let us consider the coverage of the target property:

$$T_2': \qquad G(\ (\ r_2\ \wedge\ z\ )\ \Rightarrow\ X(\ (\ g_2\ \wedge\ \neg\ g_1\ )\ U\ \neg r_2\ )\ )$$

by the achieved property:

$$A_2': \qquad G(\ (\ r_1\ \wedge\ \neg r_2\ )\ \Leftrightarrow\ X\ g_1\ )$$

and the fairness property $A_1 = GF(\ \neg r_2\ )$. We find that $A_1 \wedge A_2' \Rightarrow T_2'$ is not valid, which establishes that $T_2'$ is not covered. However, we can decompose $T_2'$ into the conjunction of:

$$T_{2a}': \qquad G(\ (\ r_2\ \wedge\ z\ )\ \Rightarrow\ X(\ g_2\ U\ \neg r_2\ )\ )$$
$$T_{2b}': \qquad G(\ (\ r_2\ \wedge\ z\ )\ \Rightarrow\ X(\ \neg\ g_1\ U\ \neg r_2\ )\ )$$

The property $T_{2b}'$ is covered by $A_2'$ and $A_1$. The coverage hole is therefore in the property $T_{2a}'$ which is a more accurate (weaker) representation than $T_2'$. □

## COVERAGE ALGORITHM

The remainder of this paper will outline the algorithms for determining a structure preserving form of the coverage gap. Our algorithm takes each formula $\mathcal{F}_\mathcal{T}$ from the target specification $\mathcal{T}$ and finds the coverage gap, $\mathcal{G}$, for $\mathcal{F}_\mathcal{T}$, with respect to the achieved specification $\mathcal{A}$. Since $\mathcal{A}$ is required to cover every property in $\mathcal{T}$ (by definition), we use this natural decomposition of the problem.

**Algorithm 1 Find_Coverage_Gap($\mathcal{F}_\mathcal{T}$,$\mathcal{A}$)**

1. Compute $\mathcal{U} = \mathcal{F}_\mathcal{T} \vee \neg\mathcal{A}$
2. If $\mathcal{U}$ is not valid then
   (a) Unfold $\mathcal{U}$ up to its fixpoint to create a set of uncovered minterms, $\mathcal{U}_M$;
   (b) Use universal abstraction to eliminate signals belonging to $\mathcal{AP}_A - \mathcal{AP}_T$,
   (c) $\mathcal{F}_\mathcal{U}$ = Call Push_MT($\mathcal{F}_\mathcal{T}$, $\mathcal{U}_M$, 1);
   (d) $\mathcal{G}$ = Call Relax_TargetSpec($\mathcal{F}_\mathcal{U}$);
3. Return $\mathcal{G}$;

The first step computes the coverage gap $\mathcal{U}$ using Theorem 2. If $\mathcal{U}$ is valid then $\mathcal{F}_\mathcal{T}$ is covered. Otherwise we determine a structure preserving representation of the coverage hole. The second step of the algorithm performs this task. This step is further divided into four steps. The following subsections describe each of these steps with examples and correctness proofs.

### Step 2(a): Unfolding of $\mathcal{U}$

In this step, we recursively unfold the property $\mathcal{U}$ upto its fixpoint [4]. We then convert the unfolded property into disjunctive form and select those terms that are free from the $G$ (always), $F$ (eventually) and $U$ (until) operators and put them in the set $\mathcal{U}_M$. In other words, we consider minterms with Boolean variables and the $X$ operator. The following example demonstrates one such unfolding.

**Example 6** Consider the property $p\ U\ (q \vee r)$. After one step of unfolding, the property looks like:

$$(\ q\ \vee\ r\ )\ \vee\ (\ p\ \wedge\ X\ (\ p\ U\ (q \vee r)\ )\ )$$

Since the subformula within the scope of the $X$ operator is identical to the original property, we have reached the fixpoint.

After this unfolding we convert the property into disjunctive form and select the terms that are free of temporal operators except $X$. In this case we have two minterms, namely $q$ and $r$. □

**Theorem 3** *The property represented by the set of minterms $\mathcal{U}_M$ closes the coverage hole for $\mathcal{F}_\mathcal{T}$.* □

It is possible that $\mathcal{U}_M$ contains variables belonging to $\mathcal{AP}_A - \mathcal{AP}_T$. In order to obtain the uncovered target specification, we need to eliminate these variables. This is done in the following step.

### Step 2(b): Abstraction

In this step we universally eliminate the variables in $\mathcal{AP}_A - \mathcal{AP}_T$ from the property represented by $\mathcal{U}_M$. The following theorem establishes that after the abstraction, $\mathcal{U}_M$ still closes the coverage hole.

**Theorem 4** *The property represented by the set of minterms $\mathcal{U}_M$ after universal abstraction closes the coverage hole for $\mathcal{F}_\mathcal{T}$.* □

### Step 2(c): Distribution of the minterms

We have shown in the previous section that the set of minterms in $\mathcal{U}_M$ represents a property that closes the coverage gap between $\mathcal{F}_\mathcal{T}$ and $\mathcal{A}$. The next objective is to represent this coverage gap as a set of properties that are structurally similar to $\mathcal{F}_\mathcal{T}$. We achieve this objective by distributing the minterms in $\mathcal{U}_M$ into the structure of $\mathcal{F}_\mathcal{T}$. The following theorem shows that this method is theoretically sound.

**Theorem 5** *The property $\mathcal{U}_M \vee \mathcal{F}_\mathcal{T}$ is at least as weak as $\mathcal{U}_M$ and closes the coverage gap for $\mathcal{F}_\mathcal{T}$.* □

The remainder of this section presents the methodology for distributing the minterms in $\mathcal{U}_M$ into the structure of $\mathcal{F}_\mathcal{T}$. The intuitive idea is to push the minterms to the subformulas having similar variables. However, $\mathcal{U}_M$ may contain some minterms that contain variables from $\mathcal{AP}_\mathcal{T}$ other than those in $\mathcal{F}_\mathcal{T}$. Let us denote these variables by $\mathcal{EV}$ (for *entering variables*).

The Function *Push_MT($\mathcal{F}$, $\mathcal{U}_M$, $\theta$)* pushes the minterms in $\mathcal{U}_M$ into the syntactic structure of property, $\mathcal{F}$. To intuitively explain the working of this function, consider a case where $\mathcal{F}$ is of the form $f \Rightarrow g$. Let $Var(f)$ and $Var(g)$ denote the set of variables in $f$ and $g$ respectively. Then we compute the universal abstraction of $\mathcal{U}_M$ with respect to $Var(g)$ and recursively push the restricted minterms (containing only variables in $Var(g)$) to $g$. We compute the universal abstraction of $\mathcal{U}_M$ with respect to $Var(f) \cup \mathcal{EV}$ and recursively push the restricted minterms to $f$. The decision to push minterms containing entering variables to the left of the implication is heuristic (but correct, since we could push them either way). In case $\mathcal{F}$ was of the form $f \wedge g$, we would have to push each minterm to both $f$ and $g$. The minterms are pushed right down to the Boolean variables, where they form Boolean formulas.

The third argument, $\theta$ of the function *Push_MT* specifies whether $\mathcal{U}_M$ should be considered in disjunction with $\mathcal{F}$ (in which case $\theta = 1$) or conjunction with $\mathcal{F}$ (denoted by $\theta = 0$). At the root level, we always have $\theta = 1$ (since we compute $\mathcal{F}_\mathcal{T} \vee \mathcal{U}_M$). However the semantics of some operators require us to recursively call *Push_MT()* with $\theta = 0$.

The functions $UABS()$ and $XABS()$ used in Push_MT are as follows:

**UABS($\varphi$, $\mathcal{SV}$)** This function takes a set of minterms, $\varphi$, and a set of variables $\mathcal{SV}$ as input and universally eliminates the set of variables given by $\mathcal{AP}_\mathcal{T} - \mathcal{SV}$ from $\varphi$. It returns the property given by the union of the abstracted set of minterms.

**XABS($\varphi$)** This function takes a set of minterms, $\varphi$, extracts those minterms that are within the scope of one or more $X$ operators, and returns these minterms after dropping the most significant X operator.

---

**Algorithm 2 Push_MT($\mathcal{F}$, $\mathcal{U}_M$, $\theta$)**

---

**case**($\mathcal{F} \equiv (f \Rightarrow g)$) :
  **if** ($\theta$ = 1) {
    Push_MT($f$, $\neg UABS(\mathcal{U}_M, Var(f) \cup \mathcal{EV})$, 0);
    Push_MT($g$, $UABS(\mathcal{U}_M, Var(g))$, 1); }
  **else** { Push_MT($f$, $\neg\mathcal{U}_M$, 1); Push_MT($g$, $\mathcal{U}_M$, 0);}

**case**($\mathcal{F} \equiv (f \vee g)$) :
  **if**($\theta$ = 1) {
    Push_MT($f$, $UABS(\mathcal{U}_M, Var(f) \cup \mathcal{EV})$, 1);
    Push_MT($g$, $UABS(\mathcal{U}_M, Var(g))$, 1); }
  **else** {Push_MT($f$, $\mathcal{U}_M$, 0); Push_MT($g$, $\mathcal{U}_M$, 0);}

**case**($\mathcal{F} \equiv (f \wedge g)$):
    {Push_MT($f$, $\mathcal{U}_M$, $\theta$); Push_MT($g$, $\mathcal{U}_M$, $\theta$);}

**case**($\mathcal{F} \equiv (\neg f)$):  Push_MT($f$, $\neg\mathcal{U}_M$, $\neg\theta$);

**case**($\mathcal{F} \equiv (X\ f)$):
    {$\mathcal{U}_{M\_X}$ = $XABS(\mathcal{U}_M)$; Push_MT($f$, $\mathcal{U}_{M\_X}$, $\theta$);}

**case**($\mathcal{F} \equiv (G\ f)$ or $(F\ f)$ or $(f\ U\ g)$): return;

**case**($\mathcal{F} \equiv p \in \mathcal{AP}_T$) :
  **if**($\theta$ = 1) Replace $p$ by $p \vee \mathcal{U}_M$; else Replace $p$ by $p \wedge \mathcal{U}_M$;

---

**Lemma 1** *The property $\mathcal{F}_\mathcal{U}$ produced by Push_MT($\mathcal{F}$, $\mathcal{M}$, $\theta$) is as strong as $\mathcal{F} \vee \mathcal{M}$ when $\theta$ = 1 and as weak as $\mathcal{F} \wedge \mathcal{M}$ when $\theta$ = 0.* □

**Theorem 6** *Push_MT($\mathcal{F}_\mathcal{T}$, $\mathcal{U}_M$, 1) returns a property $\mathcal{F}_\mathcal{U}$ that closes the coverage hole.* □

### Step 2(d): Weakening of the temporal blocks

For properties having the unbounded temporal operators, namely $G$ (always), $F$ (eventually), and $U$ (until), we use heuristics to decompose the property into weaker fragments and then return those fragments that are not covered by the achieved specification. Within the weaker fragments, we may again use Algorithm Push_MT to compute the uncovered target specification more accurately.

Our intuitive idea in this step is to systematically weaken the intermediate uncovered target specification, $\mathcal{F}_\mathcal{U}$, while ensuring that it still closes the coverage hole. One of the heuristics used in our tool is based on the observation that a property can be weakened by setting some of its variable instances to true, and setting some other variable instances to false.

After choosing a variable and performing the weakening substitution, we examine whether the weakened property still closes the coverage hole. If so, then we recursively

attempt to weaken it further. Variable substitution retains the syntactic structure of the original property, and hence the uncovered target specification produced in the end is visually comparable to the original target specification.

We also use a special algorithm for handling the coverage of invariant properties of the form $\mathcal{F}_\mathcal{T} = G(\varphi)$, which are very common in formal property specifications. The intuitive idea is as shown in Algorithm 3. Theorem 7 shows that the uncovered target specification obtained by this method closes the coverage hole.

**Algorithm 3 Coverage of Invariants**

- We compute the invariant part, $\psi$ of the achieved specification $\mathcal{A}$ (that is, we have $\mathcal{A} \Rightarrow G(\psi)$). The property $\psi$ is computed by unfolding $\mathcal{A}$.
- We then compute the minterms $\mathcal{U}_M$ of $G(\varphi) \vee \neg G(\psi)$, namely the coverage gap of $G(\varphi)$ with respect to $G(\psi)$.
- These minterms are then pushed into $\varphi$ using Push_MT to obtain the weakened property, $\varphi'$.
- We return $G(\varphi')$ as the intermediate uncovered target specification and apply Step 2(d) on the subformulas of $G(\varphi')$ to further refine the uncovered target specification.

**Theorem 7** *The property, $\mathcal{F}_\mathcal{U}$, returned by Algorithm 3 closes the coverage gap of $G(\varphi)$ with respect to $\mathcal{A}$.* □

**Example 7** Let us return to the specifications shown in Example 1 and let us consider the coverage of $T_3$ by $A_2$:

$$T_3 : \quad G(\ (\ \neg r_2\ )\ \Rightarrow\ X\ g_1\ )$$
$$A_2 : \quad G(\ (\ r_1\ \wedge\ \neg r_2\ )\ \Rightarrow\ X\ g_1\ )$$

Since both properties are invariants, we use Algorithm 3. In the first step, we compute $\psi$ as $(\ r_1\ \wedge\ \neg r_2\ )\ \Rightarrow\ X\ g_1$ The set of minterms, $\mathcal{U}_M$ obtained by unfolding $T_3 \vee \neg G\psi$ is:

$$\mathcal{U}_M = \{r_2,\ X(g_1),\ r_1 \wedge \neg r_2 \wedge \neg X(g_1)\ \}$$

We now call Push_MT to distribute the minterms in $\mathcal{U}_M$ into the parse tree of $T_3$ past the $G$ operator. The uncovered part of $T_3$ after applying Push_MT is given by:

$$G(\ (\ \neg(r_1\ \vee\ r_2)\ )\ \Rightarrow\ X\ g_1\ )$$

Let us now revisit the coverage problem of Example 5, where we examine the coverage of $T_2'$ by $A_2'$ and the fairness property $A_1 = GF(\ \neg r_2\ )$:

$$T_2' : \quad G(\ (\ r_2\ \wedge\ z\ )\ \Rightarrow\ X(\ (\ g_2\ \wedge\ \neg\ g_1\ )\ U\ \neg r_2\ )\ )$$
$$A_2' : \quad G(\ (\ r_1\ \wedge\ \neg r_2\ )\ \Leftrightarrow\ X\ g_1\ )$$

In this case also, all properties are invariants, hence we apply Algorithm 3. However, in this case Algorithm 3 fails to weaken $T_2'$ and returns $T_2'$. Therefore, we systematically weaken $T_2'$ and search for weaker properties that still close the coverage gap. In this case, we find that substituting 0 for $g_1$ in $T_2'$ gives us the property:

$$T_{2a}' : \quad G(\ (\ r_2\ \wedge\ z\ )\ \Rightarrow\ X(\ g_2\ U\ \neg r_2\ )\ )$$

which is weaker than $T_2'$, but still closes the coverage gap. By our approach, no further structure preserving weakening of $T_{2a}'$ can close the coverage gap, hence we report $T_{2a}'$ as the uncovered part of $T_2'$. □

**Theorem 8** *The coverage gap $\mathcal{G}$ returned by the function Find_Coverage_gap( $\mathcal{F}_\mathcal{T}$, $\mathcal{A}$) closes the coverage hole.* □

## THE SPECMATCHER TOOL

We have tested the proposed methodology by developing a prototype tool called *SpecMatcher*. We have examined the performance of our tool on several test cases having LTL specifications, and specifically on one industry standard specification. The results are quite promising. In all of these tests, the tool produced temporal properties that closed the coverage gap and are syntactically similar to target specification properties. The runtimes were negligible (less than a second on a 2.4 GHz Pentium-4).


### Acknowledgements

The authors acknowledge Intel Corporation for partial support of this work. Pallab Dasgupta and P.P.Chakrabarti further acknowledge the partial support of the Department of Science & Technology, Govt. of India.


## REFERENCES


[1] Armoni, R., *et al*, Enhanced Vacuity Detection in Linear Time Logic. In *CAV'2003*, LNCS 2725, 368-380, 2003.

[2] Basu, P., Das, S., Dasgupta, P., Chakrabarti, P.P., Rama Mohan, C., Fix, L., Formal Verification Coverage: Are the RTL-Properties Covering the Design's Architectural Intent? In *DATE'2004*, Paris, 668-669, 2004.

[3] Chockler, H., Kupferman, O., Vardi, M.Y., Coverage Metrics for Temporal Logic Model Checking, In *Proc. of TACAS*, LNCS 2031, pp. 528-542, 2001.

[4] Clarke, E.M., Grumberg, O., and Peled, D.A., *Model Checking*, MIT Press, 2000.

[5] Hoskote, Y., Kam, T., Ho, P, Zhao, X. Coverage Estimation for symbolic model checking. In *Proc. of DAC*, 300-305, 1999.

[6] Katz, S., Geist, D., Grumberg, O., Have I written enough properties? A method of comparison between spec.and implementation. In *Proc. of CHARME*, LNCS 1703, 1999.

[7] Pnueli, A., The temporal logics of programs. In *Proc. of FOCS'1997*, 46-57, 1997.