Proof-Based Coverage Metrics for Formal Verification

Elaheh Ghassabani, Michael Whalen, Mats Heimdahl
Department of Computer Science & Engineering
University of Minnesota
MN, USA
ghass013, mwwhalen, heimdahl@umn.edu

Andrew Gacek, Lucas Wagner
Rockwell Collins
Advanced Technology Center
IA, USA
andrew.gacek, lucas.wagner@rockwellcollins.com

Abstract—When using formal verification on critical software, an important question involves whether we have specified enough properties for a given implementation model. To address this question, coverage metrics for property-based formal verification have been proposed. Existing metrics are usually based on mutation, where the implementation model is repeatedly modified and re-analyzed to determine whether mutant models are “killed” by the property set. These metrics tend to be very expensive to compute, as they involve many additional verification problems.

This paper proposes an alternate family of metrics that can be computed using the recently introduced idea of Inductive Validity Cores (IVCs). IVCs determine a minimal set of model elements necessary to establish a proof. One of the proposed metrics is both rigorous and substantially cheaper to compute than mutation-based metrics. In addition, unlike the mutation-based techniques, the design elements marked as necessary by the metric are guaranteed to preserve provability. We demonstrate the metrics on a large corpus of examples.

Keywords—coverage; requirements completeness; formal verification; inductive proofs; inductive validity cores;

I. INTRODUCTION

In safety critical systems development, an important question involves whether the requirements and testing process are adequate for the implementation. For example, in DO178B/C [30], tests are derived from requirements and the adequacy of the tests is measured by examining coverage of the implementation. As the criticality of the software increases, more rigorous adequacy measures (statement, decision, MC/DC) are required, with the justification that code must be more rigorously executed by tests to demonstrate its compliance. If complete coverage is not achieved, analysis is performed to determine whether additional tests or additional requirements are necessary to achieve coverage.

For critical systems, it has been argued that formal methods should be applied to gain higher assurance than is possible with testing [19], [27], [31]. Unfortunately, proof-based approaches tend not to answer the question as to whether implementations have functionality that is not covered by requirements. Unlike testing, for proof the whole system is ‘executed’, but many parts of the system may be irrelevant to the property that is proved.

Relatively recently, techniques have been devised for analyzing adequacy of requirements against formal implementation models specified as transition systems or Kripke structures [5], [7], [10], [15]. The mechanism used is based on mutation and proof: is it possible to prove that the requirements still hold of the system after mutating the model in some way? If so, then the requirements are incomplete with respect to the mutated part of the model. Unfortunately, previous approaches to computing coverage metrics can underapproximate which portions of a program are necessary to prove the requirements. In addition, the mutation-based analyses tend to be computationally very expensive because there are many possible mutant models to verify. For example, for model checkers, the state of the art techniques have runtimes of (in the best case) several times more than is required for proof [4].

We wish to have a graduated set of metrics, suitable for use in certification, for checking the adequacy of requirements against an implementation model that: (1) can be applied early and throughout a development cycle on different implementation artifacts, (2) are proof preserving: the portion of the implementation that is identified as covered demonstrates the fulfillment of the requirement but does not contain irrelevant information, and (3) are efficient to compute. Towards this end, we propose measures of requirements adequacy based on minimal proofs of requirements. We measure the adequacy of a set of requirements by examining an (approximately) minimal set of model elements necessary to construct a proof of all the requirements.

Our approach is applicable to reactive software that does a bounded amount of computation in response to an input for an unbounded sequence of inputs. This set contains most embedded and especially safety-critical software, and can be described as transition systems (equivalently, infinite-state “circuits” that compute next states given an input and current state). It is implemented using Inductive Validity Cores (IVCs) [13] for transition systems, and integrated into a branch of the JKind model checker [1]. In our benchmarks, we demonstrate that one of the proof-based metrics is considerably more computationally tractable than previous approaches based on mutation, averaging 24% overhead over model-checking alone, rather than the 2369% overhead required for the state of the art of the mutation-based metrics. Thus, the contributions of this

This work was carried out within the HACMS and SOSITE Phase II grants (DARPA FA8750-12-9-0179 and FA8650-16-C-7656).
node asw(alt1, alt2: int; inhibit: bool) returns (doi_on: bool);

var
  a1_below, a2_below, a1_above, a2_above,
  below, above_hyst, d1, d2: bool;

let
  a1_below = (alt1 < THRESHOLD);       // (1)
  a2_below = (alt2 < THRESHOLD);       // (2)
  a1_above = (alt1 >= T_HYST);         // (3)
  a2_above = (alt2 >= T_HYST);         // (4)
  below = a1_below or a2_below;        // (5)
  above_hyst = a1_above and a2_above;  // (6)
  doi_on = if (below and not inhibit) // (7)
           then true else d1;
  d1 = if (inhibit or above_hyst) // (8)
       then true else d1;
  d2 = (false -> pre(doi_on));        // (9)

end;

Fig. 1. Altitude Switch Model

work are:
1) A family of coverage metrics for formal verification based on minimal Inductive Validity Cores (MIVCs). Most of the proposed metrics are proof preserving.
2) A discussion of the relationship between proof-based metrics and mutation-based metrics, including a proof of equivalence between non-deterministic mutation coverage and one of the proof-based metrics (MUST-COV).
3) An implementation that efficiently computes property coverage over symbolic transition systems,
4) An initial experiment that compares our technique against a state of the art mutation-based notion of completeness.

II. Running Example

We use a very simple system from the avionics domain to illustrate our approach. An Altitude Switch (ASW) is a hypothetical device that turns power on to another subsystem, the Device of Interest (DOI), when the aircraft descends below a threshold altitude and turns the power off again after we ascend over the threshold plus some hysteresis factor. An implementation of an ASW containing two altimeters written in the Lustre [17] language (simplified and adapted from [20]) is shown in Figure 1. If the system is not “inhibited” by the user and either altimeter is below the constant THRESHOLD, then it turns on the DOI; else, if the system is inhibited or both altimeters are above the threshold plus the hysteresis factor (THRESHOLD + HYST T_Hyst), then the DOI is turned off, and if neither condition holds, then in the initial computation it is false and thereby retains its previous value. The notation (false -> pre(doi_on))) in equation (9) describes an initialized register in Lustre: in the initial state, the expression is false, and thereafter it is the previous value of doi_on. In the remainder of the paper, we will use this model to illustrate aspects of requirements completeness.

III. Preliminaries

This section presents formalizations of transition systems, inductive validity cores, and background information on mutation-based coverage metrics. Although we focus the formalism below on safety properties, the approach is able to handle liveness properties through reduction to safety properties, as is performed by, e.g., K-liveness [8].

A. Models, Requirements, and Provability

Given a state space U, a transition system (I, T) consists of an initial state predicate I : U → bool and a transition step predicate T : U × U → bool. We define the notion of reachability for (I, T) as the smallest predicate R : U → bool which satisfies the following formulas:

∀u, I(u) ⇒ R(u)
∀u, u'. R(u) ∧ T(u, u') ⇒ R(u')

A safety property P : U → bool is a state predicate that holds on a transition system (I, T) if it holds on all reachable states, i.e., ∀u. R(u) ⇒ P(u), written as R ⇒ P for short. When this is the case, we write (I, T) ⊢ P. We assume the transition relation has the structure of a top-level conjunction. This assumption gives us a structure that we can easily manipulate. Given T(u, u') = T₁(u, u') ∧ ··· ∧ Tₙ(u, u') we will write T = T₁ ∧ ··· ∧ Tₙ for short. By further abuse of notation, T is identified with the set of its top-level conjuncts. Thus, x ∈ T means that x is a top-level conjunct of T, and S ⊆ T means all top-level conjuncts of S are top-level conjuncts of T. When a top-level conjunct x is removed from T, it is written as T \ {x}. Such a transition system can easily encode our example model in Section II. We assume each equation defines a conjunct within the transition system which we will denote by the variable assigned, so T = {a1_below, a2_below, a1_above, a2_above, below, above_hyst, d1, d2}.

Definition 1. Inductive Validity Core (IVC) [13]: S ⊆ T for (I, T) ⊢ P is an Inductive Validity Core, denoted by IVC(P, S), iff (I, S) ⊢ P.

In examining provability, we are interested in minimal sets that satisfy a property P; tracing a property to the entire model is not particularly enlightening. Fortunately, IVCs have the following monotonicity property [13]: given (I, T) ⊢ P, ∀S₁ ⊆ S₂ ⊆ T. IVC(P, S₁) ⇒ IVC(P, S₂). We next introduce the notion of minimal inductive validity cores.

Definition 2. Minimal Inductive Validity Core (MIVC) [13]: S ⊆ T is a minimal Inductive Validity Core, denoted by MIVC(P, S), iff IVC(P, S) ∧ ∀Tᵢ ∈ S. (I, S \ {Tᵢ}) ⊭ P.

Note that given (I, T) ⊢ P, P always has at least one MIVC, which implies MIVCs are not necessarily unique. For example, take I = a ∧ h, T = a’ ∧ b’, and P = a ∨ h. Then both {a’} and {b’} are MIVCs for (I, T) ⊢ P. To capture this fact, the all MIVCs (AIVC) relation has been introduced [29].

AIVC(P) ≡ { S | S ⊆ T ∧ MIVC(P, S)}

Establishing the AIVC for a single property, one gets a clear picture of the all the model elements constrained by the property. The set of AIVCs for all properties demonstrates a complete mapping from the requirements to the design elements, which is called complete traceability [29].
B. Coverage and Mutations

In general, specification completeness can be defined with regard to the notion of coverage. In fact, the way that coverage is formalized plays a key part in the strength/effectiveness of a method for the assessment of completeness. The goal of a coverage metric is usually to assign a numeric score that describes how well properties cover the design. The majority of the work on coverage metrics has focused on mutations, which are “atomic” changes to the design, where the set of possible mutations depends on the notation that is used. A mutant is “killed” if one of the properties that is satisfied by the original design is violated by the mutated design [4]–[6], [24], [25]. There are many different kinds of mutations that have been proposed, primarily focused on checking sequential bit-level hardware designs. Similarly, for software, it is possible to apply any of the “standard” source code mutation operators used for software testing [3] towards requirements coverage analysis.

We assume each element $T_i \in T$ has a set of possible mutations associated with it. Depending on the modeling formalism used, this may be the value of a gate or signal or an expression within a statement in a program. We will further assume the existence of a mutation function $f_m$ that, given a model element, will return a finite set of mutations for that element. We can then define the set of mutant models $M$ as follows:

$$M = \{(T \setminus \{T_i\}) \cup \{m\} \mid T_i \in T, m \in f_m(T_i)\}$$

and then define the mutation score for property $P$ in the standard way:

**Definition 3.** Generalized mutation coverage.

$$\text{MUTANT-COV} = \frac{|\{m \mid m \in M \land (I,m) \not\models P\}|}{|M|}$$

Note that without loss of generality, we consider a single property $P$, which can be viewed as the conjunction of all the properties of the model.

In our example in Figure 1, applying the software mutations from [3] would involve manipulating the constants used in the definitions of $a1\_below$, $a2\_below$, $a1\_above$, $a2\_above$, swapping or and and in the definition of below, above\_hyst, or negating the conditions in the if/then/else statements. Even for this small model, there are many possible mutations (57). The number of single-mutation programs is roughly the product of the leaf elements of the program abstract syntax tree (AST) and the size of the chosen set of mutations, which can lead to an impractical number of verification problems.

Mutations for hardware are discussed in [4], [24], [25]. The state of the art of mutation-based coverage can be found in [4], where a design is considered as a net-list with nodes of types $\{\text{AND, INVERTER, REGISTER, INPUT}\}$. Each mutant design changes the type of a single node to INPUT. When property $\phi$ satisfied by the original net-list fails on the mutant design, it is said that a mutant is discovered for $\phi$. Then, the coverage metric for $\phi$ is defined as the fraction of the discovered mutants, based on which the coverage of a set of properties is measured as the fraction of mutants discovered by at least one property. To decrease the cost of computation, coverage analysis is performed at several stages; first, all the nodes that do not appear in the resolution proof of a given property are marked as not-covered, and the rest of the nodes are marked as unknown. Then, for the unknown nodes, the basic mutation check is performed: if a corresponding mutant design violates the property, it will be considered as covered.

IV. Proof-Based Metrics

We propose a new approach for measuring property completeness based on proof rather than mutation. We first define notation, then describe different possible metrics given a set of minimal proofs.

**Definition 4.** IVC coverage (IVC-Cov):

Given $S \in AIVC(P)$, $T_i$ is covered by $P$ via $S$ iff $T_i \in S$.

We call Definition 4 a proof-preserving metric because, with a set of the model elements marked as covered by IVC-Cov, $P$ is provable. Other notions, as will be discussed in Section IV-A, may yield subsets of the model that are insufficient to reconstruct the proof of the property. The coverage score for IVC-Cov can be calculated with:

$$\frac{|S|}{|T|}$$

Because $P$ may have multiple $MIVCs$, IVC-Cov metric can lead to various scores that belong to the following set:

$$\left\{ \frac{|S|}{|T|} \mid S \in AIVC(P) \right\}$$

Note that if an $MIVC$ contains all model elements (i.e., the model is completely covered), then there is only one possible $MIVC$, so in this case there is no diversity of scores.

Given all proofs of a particular property, it is possible to define additional, complementary coverage notions. To do so, we use the following categorization of the model elements based on $MIVC$ and $AIVC$ relations for $P$:

- $\text{MUST}(P) = \bigcap AIVC(P)$
- $\text{MAY}(P) = \left( \bigcup AIVC(P) \right) \setminus \text{MUST}(P)$
- $\text{IRR}(P) = T \setminus \bigcup AIVC(P)$

This categorization helps to identify the role and relevance of each design element in satisfying a property. Function $\text{MUST}$ specifies the parts of the model absolutely necessary for the property satisfaction. Any change to these parts will affect provability of the property. On the other hand, any single element in $\text{MAY}(P)$, may be modified without affecting satisfaction of $P$(though modifying multiple elements may require re-proof).

The $\text{IRR}$ denotes model elements that are irrelevant to the validity of $P$ [29]. Using the notions of $\text{MAY}$ and $\text{MUST}$, we can introduce additional coverage metrics.

**Definition 5.** (MAY-Cov): $T_i \in T$ is covered by $P$ iff $T_i \in \text{MAY-Cov}(P)$, where $\text{MAY-Cov}(P) = \{T_i \mid \exists S \in AIVC(P) : T_i \in S\}$. 
Definition 6. (MUST-Cov): \( T_i \in T \) is covered by \( P \) iff \( T_i \in \text{MUST-Cov}(P) \), where \( \text{MUST-Cov}(P) = \{ T_i | \forall S \in AIVC(P), T_i \in S \} \).

The MAY-Cov notion aims to deal with the fact that a property \( P \) may have several distinct MIVCs. In such cases, IVC-Cov only looks at an arbitrary MIVC which may contain a subset of MAY\((P)\), which means, depending on which MIVC it considers, every time it may report a different part of MAY\((P)\) as uncovered. However, MAY-Cov resolves this issue reporting the entire set of MAY\((P)\) as covered, which also leads to higher coverage scores. MUST-Cov takes the opposite view, considering a model element as covered only if it affects all the proofs of \( P \).

It is still possible to build more relaxed coverage metrics in which coverage is captured by looking at individual properties, rather than their conjunction. We can, for example, describe a view, considering a model element as covered only if it affects additional properties.

Lemma 2. Given a set of properties \( \Delta \) over \( T \), \( T_i \in T \) is covered iff \( T_i \in \text{Model-Cov}(T) \), where \( \text{Model-Cov}(T) = \{ T_i | \exists P \in \Delta, S \in AIVC(P), T_i \in S \} \).

A. Discussion

Based on the categorization of elements, we will state some relationships about MIVCs so to compare different proof-based metrics proposed earlier.

Lemma 1. If \( MAY(P) \neq \emptyset \), then \( P \) is not provable by \( \text{MUST}(P) \).

Now we focus on the relationship between non-deterministic mutation-based coverage and proof-based metrics. In Chockler et. al. [4], each mutant design changes the type of a single node to INPUT. Given a suitable encoding of the netlist, assigning a “fresh” input is an isomorphic operation to simply removing a \( T_i \) from \( T \). The mapping is as follows: the netlist becomes a conjunction of equations, where each vertex becomes a variable \( v_i \in U \), and where each non-input vertex becomes an assignment equation \( T_i \in T \). For example, given an AND-vertex \( v_i \) with three input edges from other vertexes \( \{ v_a, v_b, v_c \} \), we would define an equation \( T_i \in T \) of the form \( v_i = \overline{v_a} \land v_b \land \overline{v_c} \).

Given this encoding, we can reframe the non-deterministic coverage proposed in [4] as follows:

Definition 8. Nondeterministic coverage (alternate specification) (NONDET-Cov\(^*\)) [4]. \( T_i \in T \) is covered by property \( P \) iff \( T_i \in \text{NONDET-Cov}\(^*\)(P) \), where \( \text{NONDET-Cov}\(^*\)(P) = \{ T_i | \overline{\{ T \}} \land P \land \overline{\{ T_i \}} \} \land P \).

Given this definition, it becomes straightforward to define some additional properties.

Lemma 2. \( T_i \in \text{NONDET-Cov}\(^*\)(P) \Leftrightarrow T_i \in \text{MUST-Cov}(P) \).

In light of Lemma 2, the NONDET-Cov\(^*\) coverage score of specification \( P \) can be also calculated by

\[
\frac{|\text{MUST}(P)|}{|T|}
\]

Immediate from Lemmas 1 and 2, NONDET-Cov\(^*\) is not proof-preserving, while IVC-Cov is proof-preserving by definition. Note that one can define many more proof-based coverage metrics based on the MIVC/AIVC idea. Metrics that make use of the AIVC relation are computationally more expensive to compute than IVC-Cov although they might be easier to satisfy (i.e., result in higher coverage scores). In general, when coverage scores of a given property are obtained from the proposed metrics, the following relationship holds:

\[
\text{NONDET-Cov}^* \leq \text{IVC-Cov} \leq \text{MAY-Cov} \leq \text{MUST-Cov}
\]

IVC-Cov and NONDET-Cov\(^*\) are equivalent when all elements within the model are covered: if all model elements are MUST elements, then there can only be one MIVC, and this MIVC uses all of the model elements. In the implementation and experiments, we will focus on the IVC-Cov and NONDET-Cov\(^*\) metrics. Both metrics are fairly rigorous and can be computed reasonably efficiently. The equivalence of MUST-Cov and NONDET-Cov\(^*\) allows us to compare our algorithms against state-of-the-art mutation based coverage.

V. ILLUSTRATION

We illustrate the idea of completeness metrics and the “score” from different metrics on our altitude switch (ASW) example from Section II. The ASW is responsible for turning on and off a device of interest, so we formulate two requirements that describe when the ASW should be on and when it should be off. The first attempt at formalization (property set 1) is as follows:

\[
\begin{align*}
\text{on}_p &= (\text{a1}_\text{below} \text{ and } \text{a2}_\text{below}) \text{ and not inhibit} \Rightarrow \text{doi}_\text{on} = \text{true}; \\
\text{off}_p &= (\text{a1}_\text{above} \text{ and } \text{a2}_\text{above}) \text{ and inhibit} \Rightarrow \text{doi}_\text{on} = \text{false}; \\
\text{all}_p &= \text{on}_p \text{ and } \text{off}_p;
\end{align*}
\]

For each of the IVC-Cov, MAY-Cov, and MUST-Cov metrics, \( \text{all}_p \) only requires \{below, d1, doi_on\}. This small set of elements is due to a classic specification problem: using computed variables as the antecedents of implications. We therefore modify our properties to use inputs and constants as antecedents and derive:

\[
\begin{align*}
\text{on}_p &= ((\text{alt1} < \text{THRESHOLD}) \text{ and } (\text{alt2} < \text{THRESHOLD})) \\
\text{off}_p &= ((\text{alt1} > \text{T_HYST}) \text{ and } (\text{alt2} > \text{T_HYST})) \text{ and inhibit} \Rightarrow \text{doi}_\text{on} = \text{false};
\end{align*}
\]

In this version, distinctions emerge between the metrics. \( \text{all}_p \) has two MIVCs: \{a1_below, below, doi_on, d1\}, \{a2_below, below, doi_on, d1\}, because of the on_p property: in the implementation, the DOI is turned on when either of the altimeters is below the threshold, while our property states that they both must be below. The MUST elements remain the same: \{below, doi_on, d1\}, because
IVC-C slicing is, on average, 20.5x higher than the justified in our benchmarks, where coverage reported by static that slicing is too loose about the adequacy of the property set. As a baseline, we metric should be justified and yield meaningful information model required by each of the coverage metrics. Any proposed metric is efficient and the resulting set of elements does not always lead to a proof. Additional study is necessary to determine the best circumstances for use of each metric.

VI. IMPLEMENTATION AND INITIAL RESULTS

For implementation we have made use of the technique proposed by Ghassabani et al. [13] using the Algorithm IVC_UC that computes an approximately minimal IVC in an efficient way. To compute the must set of a given property as efficiently as possible, we use the algorithm from [4], adapted to use the more scalable PDR [11] algorithm rather than interpolation for model checking. We call this algorithm MUST-COV. These algorithms have been implemented in the JKind model checker [1], [9], [12]. JKind proves safety properties using multiple cooperative engines in parallel including k-induction [32], property directed reachability (PDR) [11], and template-based lemma generation [22].

To examine the efficiency and effectiveness of our approach, we ran the MUST-COV and the IVC-Cov metric against a benchmark suite consisting of 475 Lustre models (395 from [16] and 80 industrial models derived from [28] and other sources). Most of the benchmark models from [16] are small (10kB or less, with 6-40 equations) and the industrial models each are ≥80kB with over 600 equations. The overhead of the IVC_UC algorithm is, on average, 24% over the baseline proof, as opposed to the 2369% runtime overhead for the MUST-COV computation. In many cases, the IVC-Cov metric is efficient enough to run as part of the verification process.

To examine effectiveness, we examined the portion of the model required by each of the coverage metrics. Any proposed metric should be justified and yield meaningful information about the adequacy of the property set. As a baseline, we use backwards static slicing, which is used in the hardware verification community for measuring adequacy. Our claim is that slicing is too loose a metric to be useful. This claim was justified in our benchmarks, where coverage reported by static slicing is, on average, 20.5x higher than the IVC-Cov metric.

Our metrics have a justification from proof: for each of the metrics, any model element returned must be necessary to some proof. However, there are differences between the reported coverage of the metrics. On average, the IVC-Cov metric yields 1.5x higher coverage than the MUST-COV metric. This might lead one to prefer the MUST-COV metric, but it is more expensive to compute and the resulting set of elements does not always lead to a proof. Additional study is necessary to determine the best circumstances for use of each metric.

VII. RELATED WORK

Coverage in verification was introduced in [21], [23]. Hoskote et al. [21] suggested a state-based metric in model checking based on FSM mutations, which are small atomic changes to the design. Later in [6], Chockler et al. provided corresponding notions of metrics used in simulation-based verification for formal verification. However, the proposed algorithm in [6] is both computationally expensive (approximately linear in the number of mutations). Most of the mutation-based metrics, including [5], [25], are focused on finite state systems and hardware systems. A more recent work in [4] performs coverage analysis through interpolation [26]. This work is also based on design-dependent mutations [6], as explained in III-B. The proposed algorithm is similar to the IVC-MUST algorithm as discussed in IV-A except that our implementation uses PDR and k-induction rather than interpolation.

A similar notion to ours was outlined in a patent [18], which sketches a family of proof core-based metrics for hardware verification. While the approach described by the patent is general, no formal description of the models, metrics, algorithms, or experimental results are provided.

VIII. CONCLUSIONS & FUTURE WORK

In this paper, we have examined the use of proof-based coverage notions for formal verification. These provide an alternate way of measuring property completeness, which, in some cases, are much more efficient than existing work in the literature while still providing accurate information about the covered parts of a given design. Most of our proposed metrics preserve provability, which means that the set of elements considered covered by our algorithm is sufficient to establish the validity proof for every requirement in the set of specifications. The notion of proof preservation is appealing because it allows a concrete demonstration to the user of the irrelevance of portions of the implementation. The utility of the proposed metrics are being evaluated by Rockwell Collins on a pilot project. The proposed metrics appear to be useful for both traceability and adequacy checking [33].

Since some of the proposed metrics need to compute all IVCs, we have investigated efficient algorithms for computing all IVCs [14]. In cases where there are multiple minimal satisfying sets, the all IVCs give insight on multiple ways by which the model meets a requirement.
REFERENCES