Using Model Checking for Reducing the Cost of Test Generation

Hyoung Seok Hong[1] and Hasan Ural[2]

[1] Concordia Institute for Information Systems Engineering,
Concordia University
hshong@ciise.concordia.ca
[2] School of Information Technology and Engineering,
University of Ottawa
ural@site.uottawa.ca

Abstract. This paper presents a method for reducing the cost of test generation. A spanning set for a coverage criterion is a set of entities such that exercising every entity in the spanning set guarantees exercising every entity defined by the coverage criterion. The central notion used in constructing a minimum spanning set is subsumption relation. An entity subsumes another entity if exercising the former guarantees exercising the latter. We develop a method for finding subsumption relations which can be uniformly applied to a family of control flow and data flow oriented coverage criteria by reducing the problem of determining whether an entity subsumes another entity to the model checking problem of the linear temporal logic LTL.

1 Introduction

In structural testing, we are given a coverage criterion defining a set of entities in the structure of a program and we generate a test suite satisfying the coverage criterion. A test suite is a set of test sequences and is said to satisfy a coverage criterion if for every entity defined by the coverage criterion, there is a test sequence in the test suite exercising the entity. There are a number of coverage criteria for structural testing and most of them are based on the information of control flow and data flow. We refer the interested readers to [20, 7, 17] for surveys of coverage criteria in software testing, protocol conformance testing, and hardware testing, respectively. Control flow oriented coverage criteria call for exercising single entities such as statements and branches. Data flow oriented coverage criteria call for exercising associations between definitions and uses of variables such as definition-use pairs[16], definition-use chains of fixed length[15], definition-use chains between inputs and outputs[18, 19], and ordered definition contexts[11].

For a program and a coverage criterion, the *optimal test generation problem* consists of generating a test suite satisfying the coverage criterion with a minimum number of test sequences. In [9, 10], the authors show that the complexity

J. Grabowski and B. Nielsen (Eds.): FATES 2004, LNCS 3395, pp. 110–124, 2005.

of this problem is NP-hard. Hence approaches for reducing the cost of test generation should be heuristic. In the software testing literature, several approaches have been proposed[1, 3, 4, 5, 6, 8, 13, 14]. The main idea of these approaches is to construct a subset of entities for a coverage criterion such that exercising every entity in the subset guarantees exercising every entity defined by the coverage criterion. That is, if a test suite covers every entity in the subset, the test suite satisfies the coverage criterion. Following the terminology of [13, 14], we call the subset a *spanning set* for the coverage criterion. A minimum spanning set allows one to significantly reduce the cost of test generation by focusing only the entities in the spanning set. For example, experiments in [1] show that for all-statements and all-branches coverage criteria, the entities of minimum spanning sets are around 30% of the original entities.

The central notion used in constructing a minimum spanning set is *subsumption* relation. An entity subsumes another entity if exercising the former guarantees exercising the latter. Once we have a test sequence exercising an entity, all the entities subsumed by the entity can be safely ignored. In [1, 3, 4, 5, 6, 8, 13, 14], a number of methods have been proposed for finding subsumption relations. All of them, however, investigate only simple coverage criteria such as all-statements and all-branches coverage criteria [1, 3, 4, 5, 6, 14] and all-uses coverage criterion[8, 13, 14] and cannot be generalized to more complicated data flow oriented coverage criteria.

In this paper, we develop a method for finding subsumption relations which can be uniformly applied to various coverage criteria ranging from all-statements and all-branches coverage criteria to data flow oriented coverage criteria proposed by Rapps and Weyuker[16], Ntafos[15], Ural *et al.*[18, 19], and Laski and Korel[11]. For each coverage criterion, we reduce the problem of determining whether an entity subsumes another entity to the model checking problem of the linear temporal logic LTL[12] in a succinct and rigorous way. We associate an LTL formula with every entity defined by a coverage criterion. Each formula has the following property: a path is a test sequence exercising the entity if and only if the path satisfies the formula. As a direct consequence of this property, we have that an entity e subsumes another entity e' if and only if every path satisfies $\psi \rightarrow \psi'$, where ψ and ψ' are the LTL formulas associated with e and e', respectively.

In addition to being applicable to various coverage criteria, our method has two other distinguishing features. First, the method is language independent in that the temporal logic formulas employed in the method can be applied to various kinds of programming languages and requirements specification languages. Since all the details about algorithms and implementations for finding subsumption relations are hidden in model checkers, it is not necessary to build a dedicated tool for each language. Second, the method enables one to reduce the cost of test generation for large and complex software whose size is limited by the capabilities of current model checkers. More importantly, we can enjoy the continuing and rapid advances in the model checking literature.

The remainder of the paper is organized as follows. Section 2 recalls the basics of LTL and flow graph, which are the logic and model employed in our method, respectively. Section 3 defines spanning sets and describes how to construct a minimum spanning set. Section 4 reduces the problem of finding subsumption relations to the problem of LTL model checking, which is the main result of the paper. Finally, Section 5 concludes the paper with a discussion of future work.

2 Preliminaries

Formulas of LTL are built from a set AP of atomic propositions, the standard boolean operators, and the temporal operators $\mathbf{X}$ (next time) and $\mathbf{U}$ (until) according to the following grammar: $\psi := p \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi$ where $p \in AP$. We also use the temporal operators $\mathbf{F}$ (eventually) and $\mathbf{G}$ (always) defined by $\mathbf{F}\psi \equiv true\mathbf{U}\psi$ and $\mathbf{G}\psi \equiv \neg\mathbf{F}\neg\psi$.

A Kripke structure is a tuple $M = (Q, q_{init}, L, R)$ where Q is a finite set of states, $q_{init} \in Q$ is the initial state, L: $Q \rightarrow 2^{AP}$ is the function that labels each state with atomic propositions, and $R \subseteq Q \times Q$ is the transition relation which is total, i.e., for every state q, there exists a state q' such that $(q, q') \in R$. A path of a Kripke structure is an infinite sequence $\pi = q_0 q_1 ...$ of states such that for every $i \geq 0$, $(q_i, q_{i+1}) \in R$. For a position i, $\pi(i)$ is the i-th element of a path π and π^i is the suffix $q_i q_{i+1} ...$ of π.

For a path π and an LTL formula ψ, we write $\pi \models \psi$ to denote that π satisfies ψ. The satisfaction relation $\models$ is defined inductively as follows:

- $\pi \models p$ iff $p \in L(\pi(0))$.
- $\pi \models \neg\psi$ iff $\pi \not\models \psi$.
- $\pi \models \psi_1 \wedge \psi_2$ iff $\pi \models \psi_1$ and $\pi \models \psi_2$.
- $\pi \models \mathbf{X}\psi$ iff $\pi^1 \models \psi$.
- $\pi \models \psi_1 \mathbf{U} \psi_2$ iff there exists $i \geq 0$ such that $\pi^i \models \psi_2$ and $\pi^j \models \psi_1$ for every $0 \leq j < i$.

For a Kripke structure M and an LTL formula ψ, we write $M \models \psi$ if for every path π such that $\pi(0) = q_{init}$, $\pi \models \psi$. The model checking problem of LTL is to decide if for given M and ψ, it holds that $M \models \psi$.

A *flow graph* of a program module is a directed graph $G = (V, v_s, v_f, A)$ where V is a finite set of vertices, $v_s \in V$ is the start vertex, $v_f \in V$ is the final vertex, and $A \subseteq V \times V$ is a finite set of arcs. The start vertex v_s and final vertex v_f represent the single entry and single exit point of a program module, respectively. A vertex represents a simple statement (such as assignment, input, and output) or the condition of a conditional or repetitive statement. An arc represents possible flow of control between statements. A finite path $v_1 ... v_n$ of a flow graph is *complete* if $v_1 = v_s$ and $v_n = v_f$. A *test sequence* is a complete path and a *test suite* is a finite set of test sequences. Figure 1 shows a flow graph where v_1 is the start vertex and v_6 is the final vertex. There are two test sequences $v_1 v_2 v_3 v_4 v_5 v_6$ and $v_1 v_2 v_3 v_5 v_6$ in Figure 1.

Each variable occurrence in a program module is classified as a definition or use. A variable x is *defined* at a vertex v, denoted by $d(x, v)$, if v represents

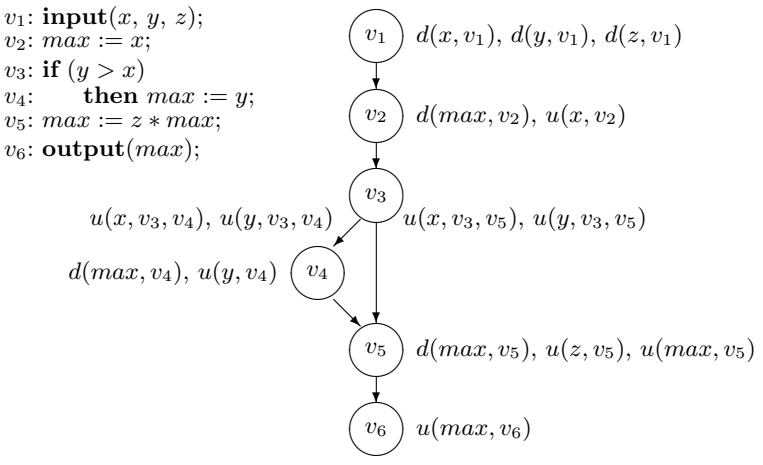


Fig. 1. An example of flow graphs

a statement assigning a value to x. A variable x is *computation-used* (c-used) at a vertex v, denoted by $u(x, v)$, if v represents a statement referencing x. A variable x is *predicate-used* (p-used) at an arc (v, v'), denoted by $u(x, v, v')$, if v represents the condition of a conditional or repetitive statement referencing x. A *use* is either a c-use or p-use.

We view a flow graph as a Kripke structure. The Kripke structure corresponding to a flow graph $G = (V, v_s, v_f, A)$ is $(V, v_s, L, A \cup \{(v_f, v_f)\})$ where $L(v) = \{v\}$ for every vertex $v \in V$. The tuple (v_f,v_f) is necessary to guarantee that the transition relation be total. We will not distinguish between flow graphs and their Kripke structures because of their simple correspondence. In addition, we will identify a test sequence $v_s...v_f$ with the infinite path $v_s...v_f v_f v_f....$

3 Spanning Sets

We adopt the following terminology introduced in [13, 14]. For a flow graph G and a coverage criterion C, $E(G, C)$ is the set of entities of G required to be exercised by C. A subset of $E(G, C)$ is a *spanning set* if exercising every entity in the subset guarantees exercising every entity in $E(G, C)$. A *minimum spanning set* is a spanning set S such that $|S| \leq |S'|$ for every spanning set S'. It is easy to see that a test suite exercises every entity in a spanning set if and only if the test suite satisfies the coverage criterion. For example, for the flow graph shown in Figure 1 and all-statements coverage criterion, we observe that $E(G, C)$ is $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $\{v_4\}$ is a spanning set for $E(G, C)$. Indeed, $\{v_4\}$ is a minimum spanning set. Consider a test suite $\{v_1v_2v_3v_4v_5v_6\}$. Since the test suite exercises v_4, it also exercises all the statements $v_1, v_2, v_3, v_4, v_5, v_6$.

The central notion used in constructing a minimum spanning set is *subsumption relation*. An entity subsumes another entity if a test sequence exercising the former also exercises the latter. Once we have a test sequence exercising an entity, we do not need to generate test sequences exercising the entities subsumed by the entity. In addition, if an entity is not subsumed by any other entities, a test sequence exercising the entity should be generated. In the next section, we will show how to find subsumption relations for various coverage criteria.

We construct a minimum spanning set using two graphs called *subsumption graph* and *reduced subsumption graph*[13, 14]. For a flow graph G and a coverage criterion C, the subsumption graph is $(E(G, C), SR)$ where SR is the subsumption relation between the entities in $E(G, C)$. Note that the subsumption relation SR is not a partial order and hence subsumption graphs may have strongly connected components. A reduced subsumption graph is a directed acyclic graph obtained by collapsing each strongly connected component of a subsumption graph into one vertex. Let $v_1, ..., v_n$ be the vertices of the reduced subsumption graph that have no incoming arcs, that is, the vertices that are not subsumed by any other vertices. Let $V_1, ..., V_n$ be the strongly connected components corresponding to $v_1, ..., v_n$, respectively. A minimum spanning set is $\{v'_1, ..., v'_n\}$ such that $v'_i \in V_i$ for every $1 \leq i \leq n$.

4 Subsumption Relations

This section addresses the problem of finding subsumption relations. Figure 2 shows an algorithm for finding subsumption graph in a generic fashion without being specific about any coverage criteria. For a flow graph G and a coverage criterion C, we first construct the set $E(G, C)$ of entities (Line 2) and in turn the set PE of pairs of entities (Line 3). For every pair (e, e') in PE, we determine whether e subsumes e' by model-checking the LTL formula $\mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ against the flow graph G, where $\mathbf{ltl}(e)$ and $\mathbf{ltl}(e')$ are the LTL formulas associated with e and e', respectively (Line 5). Theorem 1 proves the correctness of the algorithm.

INPUT: a flow graph G and a coverage criteron C
OUTPUT: the subsumption graph $(E(G, C), SR)$

1: $SR := \emptyset$;
2: construct the set $E(G, C)$ of entities of G required by C;
3: $PE := \{(e, e') \mid e, e' \in E(G, C), e \neq e'\}$;
4: **for** every pair (e, e') in PE **do**
5: model check $\mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ against G;
6: **if** $G \models \mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ **then** $SR := SR \cup \{(e, e')\}$;
7: **return** $(E(G, C), SR)$;

Fig. 2. An algorithm for finding a subsumption graph

Theorem 1. Assume that the LTL formula $\mathbf{ltl}(e)$ has the the following property: a path π is a test sequence exercising e if and only if $\pi \models \mathbf{ltl}(e)$. Then we have that e subsumes e' if and only if $G \models \mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$.

Proof. e subsumes e' if and only if for every path π, π is a test sequence exercising e implies π is a test sequence exercising e' if and only if for every path π, $\pi \models \mathbf{ltl}(e) \rightarrow \pi \models \mathbf{ltl}(e')$ if and only if for every path π, $\pi \models \mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ if and only if $G \models \mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$. □

In the above algorithm, the total number of model checking performed is $O(|E(G,C)|^2)$ both in the best case and worst case. Note that the subsumption graph $(E(G,C), SR)$ is used to identify all possible minimum spanning sets. If we are only interested in one minimum spanning set rather than all possible ones, we can significantly reduce the total number of model checking to $O(|E(G,C)|)$ in the best case using the new algorithm shown in Figure 3. It is not hard to see that the result of the new algorithm is a spanning forest of the subsumption graph $(E(G,C), SR)$. Moreover, the root nodes of the spanning forest comprise a minimum spanning set.

INPUT: a flow graph G and a coverage criteron C
OUTPUT: a spanning forest $(E(G,C), SF)$ for the subsumption graph $(E(G,C), SR)$

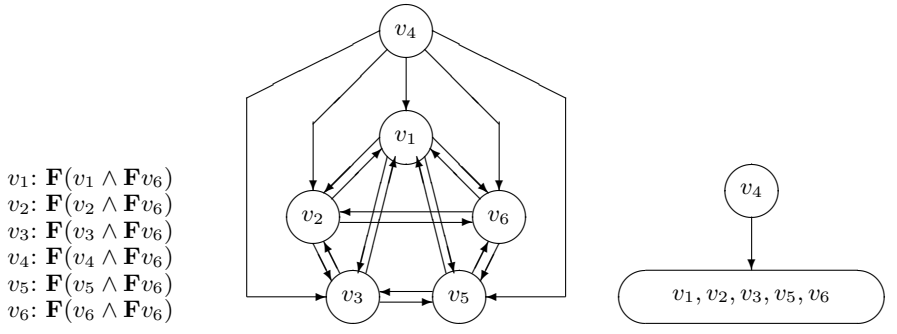
1: $SF := \emptyset$;
2: construct the set $E(G,C)$; let $E(G,C) = \{e_1, ..., e_n\}$;
3: **for** $i := 1$ to n **do** $L[i] := e_i$; *marked*[i] := *false*;
4: **for** $i := 1$ to n **do**
5: **if** *marked*[i] = *false* **then**
6: **for** $j := 1$ to n, $j \neq i$ **do**
7: **if** *marked*[j] = *false* **then**
8: model check $\mathbf{ltl}(L[i]) \rightarrow \mathbf{ltl}(\mathrm{L[j]})$ against G;
9: if $G \models \mathbf{ltl}(L[i]) \rightarrow \mathbf{ltl}(\mathrm{L[j]})$ **then**
10: $SF := SF \cup \{(L[i], L[j])\}$;
11: *marked*[j] := *true*;
12: **return** $(E(G,C), SF)$;

Fig. 3. An algorithm for finding a spanning forest of the subsumption graph

In the following sections, for each coverage criterion, for each entity e in $E(G,C)$, we will define the LTL formula $\mathbf{ltl}(e)$ and show its property that a path π is a test sequence exercising e if and only if $\pi \models \mathbf{ltl}(e)$.

4.1 Statements and Branches

All-Statements Coverage Criterion. We say that a test sequence π exercises a vertex v if there is $i \geq 0$ such that $\pi(i) = v$. A test suite Π satisfies *all-statements coverage criterion* if every vertex of a flow graph is exercised by a test sequence in Π. For a vertex v, we associate an LTL formula defined by



(a) LTL formulas (b) subsumption graph (c) reduced subsumption graph

Fig. 4. All-statements coverage criterion for Figure 1

$$\mathbf{ltl}(v) = \mathbf{F}(v \wedge \mathbf{F}v_f)$$

with the property that a path π is a test sequence exercising a vertex v if and only if there are $0 \leq i \leq j$ such that $\pi(i) \models v$ and $\pi(j) \models v_f$ if and only if $\pi \models \mathbf{F}(v \wedge \mathbf{F}v_f)$. Figure 4.(a) shows the vertices and their LTL formulas for the flow graph in Figure 1. By model-checking the formula $\mathbf{ltl}(v) \rightarrow \mathbf{ltl}(v')$ for every pair (v, v') of vertices, we obtain the subsumption graph shown in Figure 4.(b). We then collapse the strongly connected component $\{v_1, v_2, v_3, v_5, v_6\}$ into one vertex and obtain the reduced subsumption graph shown in Figure 4.(c). Finally we find a minimum spanning set $\{v_4\}$.

All-Branches Coverage Criterion. We say that a test sequence π exercises an arc (v, v') if there is $i \geq 0$ such that $\pi(i) = v$ and $\pi(i+1) = v'$. A test suite Π satisfies *all-branches coverage criterion* if every arc of a flow graph is exercised by a test sequence in Π. For an arc (v, v'), we associate an LTL formula defined by

$$\mathbf{ltl}(v, v') = \mathbf{F}(v \wedge \mathbf{X}(v' \wedge \mathbf{F}v_f))$$

with the property that a path π is a test sequence exercising an arc (v, v') if and only if there are $0 \leq i < j$ such that $\pi(i) \models v$, $\pi(i+1) \models v'$, $\pi(j) \models v_f$ if and only if $\pi \models \mathbf{F}(v \wedge \mathbf{X}(v' \wedge \mathbf{F}v_f))$. Figure 5 shows the arcs, their LTL formulas, and reduced subsumption graph for the flow graph in Figure 1. We have two minimum spanning sets $\{(v_3, v_4), (v_3, v_5)\}$ and $\{(v_4, v_5), (v_3, v_5)\}$.

4.2 Definition-Use Pairs

Rapps and Weyuker[16] propose a family of data flow oriented coverage criteria that require certain pairs between definitions and uses of the same variable be exercised. Let x be a variable, v be a vertex, and w be a vertex v' or arc (v', v'').

– A finite path $v, v_1, ..., v_n, w$ is a *definition-clear* path from v to w with respect to x if x is not defined at v_i for every $1 \leq i \leq n$.

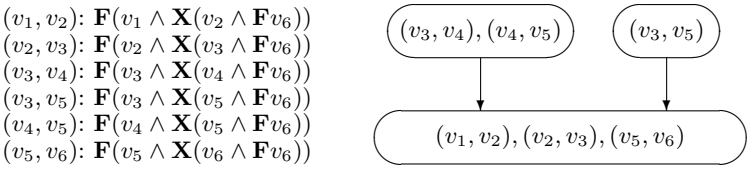


Fig. 5. All-branches coverage criterion for Figure 1

- For a definition $d(x, v)$ and use $u(x, w)$ of the same variable x, $d(x, v)$ *reaches* $u(x, w)$ if there is a definition-clear path from v to w with respect to x. If w is a vertex, the pair $(d(x, v), u(x, w))$ is called *definition-cuse pair* (dcu-pair). Otherwise, the pair is called *definition-puse pair* (dpu-pair).
- A *definition-use pair* (du-pair) is either a dcu-pair or dpu-pair.

In Figure 1, we observe that $(d(max, v_2),\ u(max, v_5))$ is a du-pair whose definition-clear path is $v_2v_3v_5$, while $(d(max, v_2),\ u(max, v_6))$ is not because there is no definition-clear path from v_2 to v_6 with respect to max.

Identifying du-Pairs. We note that the set of du-pairs of a flow graph can be identified using the conventional data flow analysis algorithm for the reaching-definition problem[2]. Recently, in [9, 10], the authors show that the set of du-pairs can also be identified using CTL model checking.

All-Uses Coverage Criterion. We say that a test sequence π exercises a du-pair $(d(x, v), u(x, w))$ if π is of the form $\pi_1 \cdot \pi_2 \cdot \pi_3$, where π_2 is a definition-clear path from v to w with respect to x. A test suite Π satisfies *all-uses coverage criterion* if every du-pair $(d(x, v), u(x, w))$ of a flow graph is exercised by a test sequence in Π. Let $def(x)$ be the disjunction of all vertices at which x is defined. For example, in Figure 1, $def(x) ::= v_1$, $def(y) ::= v_1$, $def(z) ::= v_1$, $def(max) ::= v_2 \vee v_4 \vee v_5$. For a du-pair $(d(x, v), u(x, w))$, we associate an LTL formula defined by

- if the pair is a dcu-pair, i.e., w is a vertex v',
  $\mathbf{ltl}(d(x, v), u(x, v')) = \mathbf{F}(v \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{F}v_f)])$
- if the pair is a dpu-pair, i.e., w is an arc (v', v''),
  $\mathbf{ltl}(d(x, v), u(x, v', v'')) = \mathbf{F}(v \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{X}(v'' \wedge \mathbf{F}v_f))])$

with the property that a path π is a test sequence exercising a dcu-pair $(d(x, v), u(x, v'))$ if and only if there are $0 \leq i < j \leq k$ such that $\pi(i) \models v$, $\pi(l) \models \neg def(x)$ for $i < l < j$, $\pi(j) \models v'$, and $\pi(k) \models v_f$ if and only if $\pi \models \mathbf{F}(v \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{F}v_f)])$. The same property also holds for dpu-pairs. Figure 6 shows the du-pairs, their LTL formulas, and the reduced subsumption graph for the flow graph in Figure 1.

All-Defs Coverage Criterion. For a definition $d(x, v)$, define $DUPAIR(d(x, v))$ as the set of du-pairs whose definition is $d(x, v)$. We say that a test sequence π exercises a definition $d(x, v)$ if π exercises a du-pair in $DUPAIR(d(x, v))$.

$(d(x, v_1), u(x, v_2))$: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v_2 \wedge \mathbf{F}v_6)])$
$(d(x, v_1), u(x, v_3, v_4))$: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v_3 \wedge \mathbf{X}(v_4 \wedge \mathbf{F}v_6))])$
$(d(x, v_1), u(x, v_3, v_5)))$: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v_3 \wedge \mathbf{X}(v_5 \wedge \mathbf{F}v_6))])$
$(d(y, v_1),\ u(y, v_4))$: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(y)\mathbf{U}(v_4 \wedge \mathbf{F}v_6)])$
$(d(y, v_1),\ u(y, v_3, v_4))$: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(y)\mathbf{U}(v_3 \wedge \mathbf{X}(v_4 \wedge \mathbf{F}v_6))])$
$(d(y, v_1),\ u(y, v_3, v_5))$: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(y)\mathbf{U}(v_3 \wedge \mathbf{X}(v_5 \wedge \mathbf{F}v_6))])$
$(d(z, v_1),\ u(z, v_5))$: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(z)\mathbf{U}(v_5 \wedge \mathbf{F}v_6)])$
$(d(max, v_2),\ u(max, v_5))$: $\mathbf{F}(v_2 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{F}v_6)])$
$(d(max, v_4),\ u(max, v_5))$: $\mathbf{F}(v_4 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{F}v_6)])$
$(d(max, v_5),\ u(max, v_6))$: $\mathbf{F}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])$

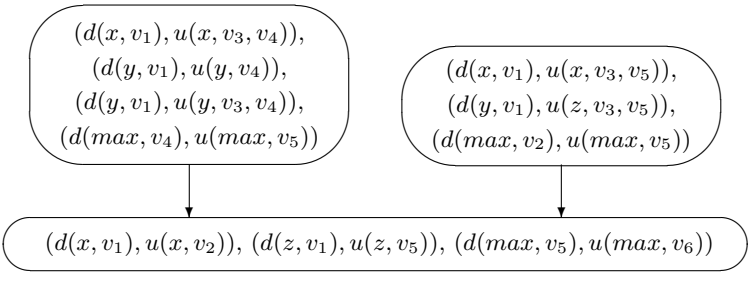


Fig. 6. All-uses coverage criterion for Figure 1

A test suite Π satisfies *all-defs coverage criterion* if every definition $d(x, v)$ of a flow graph is exercised by a test sequence in Π. For a definition $d(x, v)$, we associate an LTL formula defined by

$$\mathbf{ltl}(d(x, v)) = \bigvee_{(d(x,v),u(x,w))\in DUPAIR(d(x,v))} \mathbf{ltl}(d(x, v), u(x, w))$$

with the property that a path π is a test sequence exercising $d(x, v)$ if and only if π exercises a du-pair in $DUPAIR(d(x, v))$ if and only if $\pi \models \mathbf{ltl}(d(x, v))$. Figure 7 shows the definitions, their LTL formulas, and the reduced subsumption graph for the flow graph in Figure 1.

4.3 Required k-Tuples

Ntafos[15] emphasizes interactions between different variables. Such interactions are captured in terms of sequences of du-pairs.

- A sequence $[d(x_1, v_1), u(x_1, w_1),\ ...\ d(x_n, v_n), u(x_n, w_n)]$ of du-pairs is a *data flow chain* (df-chain)[18] if for every $1 \leq i < n$, $w_i = v_{i+1}$, that is, $u(x_i, w_i)$ and $d(x_i, v_{i+1})$ occur at the same vertex and hence the definition $d(x_i, v_{i+1})$ is given in terms of $u(x_i, w_i)$. Note that $u(x_i, w_i)$, $1 \leq i < n$, is a c-use and the final use $u(x_n, w_n)$ may be either a c-use or p-use.
- A df-chain consisting of $k - 1$ du-pairs, $k \geq 2$, is a *k-definition/reference interaction* (k-dr interaction) in the terminology of [15].

$d(x, v_1)$: $\mathbf{ltl}(d(x, v_1), u(x, v_2)) \vee \mathbf{ltl}(d(x, v_1), u(x, v_3, v_4)) \vee \mathbf{ltl}(d(x, v_1), u(x, v_3, v_5))$
$d(y, v_1)$: $\mathbf{ltl}(d(y, v_1), u(y, v_4)) \vee \mathbf{ltl}(d(y, v_1), u(y, v_3, v_4)) \vee \mathbf{ltl}(d(y, v_1), u(y, v_3, v_5))$
$d(z, v_1)$: $\mathbf{ltl}(d(z, v_1), u(z, v_5))$
$d(max, v_2)$: $\mathbf{ltl}(d(max, v_2), u(max, v_5))$
$d(max, v_4)$: $\mathbf{ltl}(d(max, v_4), u(max, v_5))$
$d(max, v_5)$: $\mathbf{ltl}(d(max, v_5), u(max, v_6))$

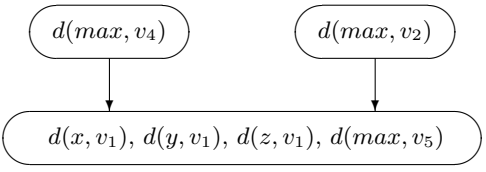


Fig. 7. All-defs coverage criterion for Figure 1

– A path $v_1\pi_1w_1...v_n\pi_nw_n$ is an *interaction path* of a df-chain if for every $1 \leq i \leq n$, $v_i\pi_iw_i$ is a definition-clear path from v_i to w_i with respect to x_i.

In Figure 1, we observe that $[d(x, v_1), u(x, v_2)]$ is a 2-dr interaction that has v_1v_2 as its interaction path and $[d(x, v_1), u(x, v_2),\ d(max, v_2), u(max, v_5)]$ is a 3-dr interaction that has $v_1v_2v_3v_5$ as its interaction path.

Identifying *k*-dr Interactions. Let $\kappa = [d(x_1, v_1),\ u(x_1, w_1),\ ...\ d(x_{k-1}, v_{k-1}),\ u(x_{k-1}, w_{k-1})]$. By definition, κ is a k-dr interaction if and only if $(d(x_1, v_1),\ u(x_1, w_1))$ is a du-pair, $w_1 = v_2$, and $[d(x_2, v_2),\ u(x_2, w_2),\ ...\ d(x_{k-1}, v_{k-1}),\ u(x_{k-1}, w_{k-1})]$ is a $(k-1)$-dr interaction. This leads to a recursive algorithm for identifying the set of k-dr interactions.

Required *k*-Tuples Coverage Criterion. We say that a test sequence π exercises a k-dr interaction κ if π is of the form $\pi_1 \cdot \pi_2 \cdot \pi_3$, where π_2 is an interaction path of κ. A test suite Π satisfies *required k-tuples coverage criterion* if every k-dr interaction of a flow graph is exercised by a test sequence in Π. For a k-dr interaction κ, $k \geq 2$, we associate an LTL formula inductively defined by

– $\mathbf{ltl}(\kappa) = \mathbf{F}ltl(\kappa)$,
– if κ is $[d(x, v), u(x, v')] \cdot \kappa'$, then $ltl(\kappa) = (v \wedge \mathbf{X}[\neg def(x)\mathbf{U}ltl(\kappa')])$,
– if κ is $[d(x, v), u(x, v')]$, then $ltl(\kappa) = (v \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{F}v_f)])$,
– if κ is $[d(x, v), u(x, v', v'')]$, then $ltl(\kappa) = (v \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{X}(v'' \wedge \mathbf{F}v_f))])$.

By induction on the number of du-pairs in κ, it can be shown that a path π is a test sequence exercising a k-dr interaction κ if and only if $\pi \models \mathbf{ltl}(\kappa)$. Figure 8 shows the 3-dr interactions, their LTL formulas, and the reduced subsumption graph for the flow graph in Figure 1.

4.4 IO-df-Chains

Ural *et al.*[18, 19] also emphasize interactions between different variables. While required k-tuples coverage criterion considers df-chains consisting of a fixed number of du-pairs, all-IO-df-chains coverage criterion in [18, 19] considers df-chains

$[d(x, v_1), u(x, v_2), d(max, v_2), u(max, v_5)]$:
$\quad \mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v_2 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{F}v_6)])])$
$[d(y, v_1), u(y, v_4), d(max, v_4), u(max, v_5)]$:
$\quad \mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(y)\mathbf{U}(v_4 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{F}v_6)])])$
$[d(z, v_1), u(z, v_5), d(max, v_5), u(max, v_6)]$:
$\quad \mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(z)\mathbf{U}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])])$
$[d(max, v_2), u(max, v_5), d(max, v_5), u(max, v_6)]$:
$\quad \mathbf{F}(v_2 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])])$
$[d(max, v_4), u(max, v_5), d(max, v_5), u(max, v_6)]$:
$\quad \mathbf{F}(v_4 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])])$

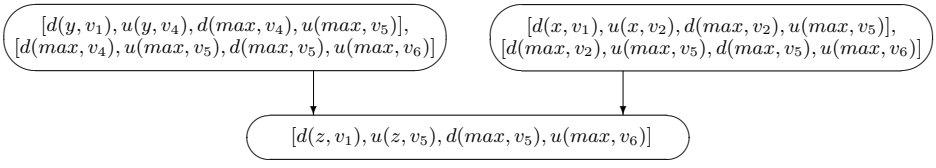


Fig. 8. 3-dr interaction coverage criterion for Figure 1

consisting of an arbitrary (but finite) number of du-pairs which start with inputs and end with outputs. In this paper, we define an *input* as a definition occurring at an input statement and *output* as a use occurring at an output statement. The rationale here is to capture the functionality of a module in terms of the interactions with its environment by identifying the effects of inputs accepted from the environment on outputs offered to the environment. Let $d(x, v)$ be a definition and $u(x', w)$ be a use.

- $d(x, v)$ *affects* $u(x', w)$ if either $x = x'$ and $(d(x, v), u(x', w))$ is a du-pair or there is a use $u(x, v')$ such that $(d(x, v), u(x, v'))$ is a du-pair and there is a definition $d(x'', v')$, given in terms of $u(x, v')$, that affects $u(x', w)$.
- $(d(x, v), u(x', w))$ is an *affect-pair* if $d(x, v)$ affects $u(x', w)$. By definition, $(d(x, v), u(x', w))$ is an affect-pair if and only if there is a df-chain $[d(x_1, v_1)$, $u(x_1, w_1)$, ... $d(x_n, v_n)$, $u(x_n, w_n)]$ such that $d(x_1, v_1) = d(x, v)$ and $u(x_n, w_n) = u(x', w)$.

Among the particular affect-pairs of interest are those starting with inputs and ending with outputs, which we call *io-pairs*. In Figure 1, there are three inputs $d(x, v_1)$, $d(y, v_1)$, $d(z, v_1)$ and one output $u(max, v_6)$. Consider the input $d(x, v_1)$ and output $u(max, v_6)$. We observe that $d(x, v_1)$ affects $u(max, v_6)$ through the df-chain $[d(x, v_1), u(x, v_2)$, $d(max, v_2), u(max, v_5)$, $d(max, v_5)$, $u(max, v_6)]$.

Identifying Simple df-Chains. For a definition $d(x, v)$ and use $u(x', w)$, we use $CHAIN(d(x, v), u(x', w))$ to denote the set of df-chains $\kappa = [d(x_1, v_1)$, $u(x_1, w_1)$, ... $d(x_n, v_n)$, $u(x_n, w_n)]$ such that $d(x_1, v_1) = d(x, v)$ and $u(x_n, w_n) = u(x', w)$. In general, there may be multiple occurrences of the same du-pair in

κ thereby causing the possibility of an infinite number of df-chains in $CHAIN(d(x,v), u(x',w))$. In order to put an upper bound on the size of $CHAIN(d(x,v), u(x',w))$, we consider its subset $SCHAIN(d(x,v), u(x',w))$ consisting of *simple* df-chains that are allowed to have at most one occurrence of each du-pair. By definition, κ is a simple df-chain in $SCHAIN(d(x,v), u(x',w))$ if and only if $d(x_1,v_1) = d(x,v)$, $u(x_n,w_n) = u(x',w)$, $(d(x_1,v_1), u(x_1,w_1))$ is a du-pair, and $[d(x_2,v_2), u(x_2,w_2)$, ... $d(x_n,v_n), u(x_n,w_n)]$ is a simple df-chain that does not contain the first du-pair $(d(x_1,v_1), u(x_1,w_1))$. This leads to a recursive algorithm for identifying the set of simple df-chains.

All-IO-df-Chains Coverage Criterion. A test suite Π satisfies *all-IO-df-chains coverage criterion* if for every io-pair (i,o), every simple df-chain in $SCHAIN(i,o)$ is covered by a test sequence in Π. For a simple df-chain κ in $SCHAIN(i,o)$, we associate the LTL formula $\mathbf{ltl}(\kappa)$. For example, in Figure 1, there are three io-pairs $(d(x,v_1), u(max,v_6))$, $(d(y,v_1), u(max,v_6))$, and $(d(z,v_1), u(max,v_6))$. Figure 9 shows the simple chains for the io-pairs, their LTL formulas, and the reduced subsumption graph.

$[d(x,v_1), u(x,v_2), d(max,v_2), u(max,v_5), d(max,v_5), u(max,v_6)]$:
$\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v_2 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])])])$
$[d(y,v_1), u(y,v_4), d(max,v_4), u(max,v_5), d(max,v_5), u(max,v_6)]$:
$\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(y)\mathbf{U}(v_4 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])])])$
$[d(z,v_1), u(z,v_5), d(max,v_5), u(max,v_6)]$:
$\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(z)\mathbf{U}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])])$

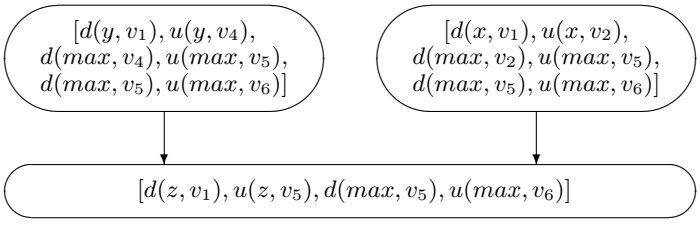


Fig. 9. All-IO-df-chains coverage criterion for Figure 1

4.5 Ordered Definition Contexts

Laski and Korel[11] emphasize that a vertex or arc may contain uses of several different variables, where each use may be reached by several different definitions. Let w be a vertex or arc and $u(x_1,w)$, ..., $u(x_n,w)$ be the uses occurring at w.

- For a set X of variables, we use $d(X,v)$ to denote the set $\{d(x,v) \mid x \in X\}$ of definitions.
- An *ordered definition context* of w is a sequence $[d(X_1,v_1), ..., d(X_m,v_m)]$ of sets of definitions such that $X_1 \cup ... \cup X_m = X$ and there is a path $v_1\pi_1...v_m\pi_m w$, called *ordered context path*, satisfying the following property: for every $1 \leq i \leq m$, $v_i\pi_i...v_m\pi_m w$ is a definition-clear path from v_i to w with respect to every variable x in X_i.

$[d(\{x\}, v_1)]$ of v_2: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v_2 \wedge \mathbf{F}v_6)])$
$[d(\{x, y\}, v_1)]$ of (v_3, v_4): $\mathbf{F}(v_1 \wedge \mathbf{X}[(\neg def(x) \wedge \neg def(y))\mathbf{U}(v_3 \wedge \mathbf{X}(v_4 \wedge \mathbf{F}v_6))])$
$[d(\{x, y\}, v_1)]$ of (v_3, v_5): $\mathbf{F}(v_1 \wedge \mathbf{X}[(\neg def(x) \wedge \neg def(y))\mathbf{U}(v_3 \wedge \mathbf{X}(v_5 \wedge \mathbf{F}v_6))])$
$[d(\{x\}, v_1)]$ of v_4: $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v_4 \wedge \mathbf{F}v_6)])$
$[d(\{z\}, v_1), d(\{max\}, v_2)]$ of v_5:
$\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(z)\mathbf{U}(\neg def(z) \wedge v_2 \wedge \mathbf{X}[(\neg def(z) \wedge \neg def(max))\mathbf{U}(v_5 \wedge \mathbf{F}v_6)])])$
$[d(\{z\}, v_1), d(\{max\}, v_4)]$ of v_5:
$\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(z)\mathbf{U}(\neg def(z) \wedge v_4 \wedge \mathbf{X}[(\neg def(z) \wedge \neg def(max))\mathbf{U}(v_5 \wedge \mathbf{F}v_6)])])$
$[d(\{max\}, v_5)]$ of v_6: $\mathbf{F}(v_5 \wedge \mathbf{X}[\neg def(max)\mathbf{U}(v_6 \wedge \mathbf{F}v_6)])$

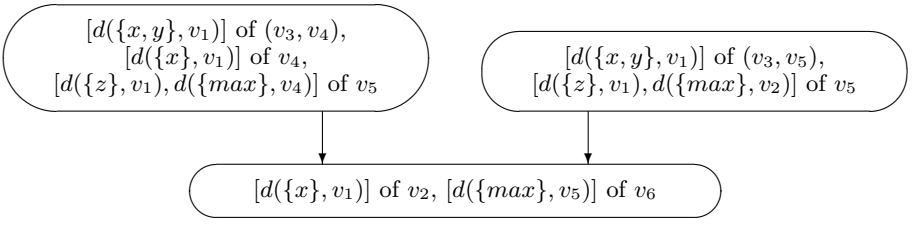


Fig. 10. Ordered contexts coverage criterion for Figure 1

In Figure 1, consider the vertex v_5 that has two uses $u(z, v_5)$ and $u(max, v_5)$. We observe that $[d(\{z\}, v_1), d(\{max\}, v_2)]$ and $[d(\{z\}, v_1), d(\{max\}, v_4)]$ are ordered definition contexts of v_5 which have $v_1v_2v_3v_5$ and $v_1v_2v_3v_4v_5$ as their ordered context path, respectively. For another example, consider the edge (v_3, v_4) that has two uses $u(x, v_3, v_4)$ and $u(y, v_3, v_4)$. $[d(\{x, y\}, v_1)]$ is an ordered definition context of the edge.

Identifying Ordered Definition Contexts. Let λ=$[d(X_1, v_1), ..., d(X_m, v_m)]$. By definition, λ is an ordered definition context of w if and only if for every $1 \leq i \leq m$,

- for every variable $x \in X_1 \cup ... \cup X_{i-1}$, x is not defined at v_i,
- for every variable $x \in X_1 \cup ... \cup X_i$, there is a definition-clear path from v_i to v_{i+1} with respect to x.

This leads to a recursive algorithm for identifying the set of ordered definition contexts.

Ordered Contexts Coverage Criterion. We say that a test sequence π exercises an ordered definition context λ if π is of the form $\pi_1 \cdot \pi_2 \cdot \pi_3$, where π_2 is an ordered context path of λ. A test suite Π satisfies *ordered contexts coverage criterion* if for every vertex or arc w of a flow graph, every ordered definition context of w is exercised by a test sequence in Π. For an ordered definition context λ of w, we associate an LTL formula inductively defined by

- $\mathbf{ltl}(\lambda) = \mathbf{F}ltl(\lambda, true)$,
- if λ is $[d(X, v)]\cdot\lambda'$, then $ltl(\lambda, nodef) = (nodef \wedge v \wedge \mathbf{X}[nodef'\mathbf{U}ltl(\lambda', nodef')])$, where $nodef' = nodef \wedge \bigwedge_{x \in X} \neg def(x)$,

– if λ is empty and $w = v'$, then $ltl(\lambda, nodef) = (v' \wedge \mathbf{F}v_f)$,
– if λ is empty and $w = (v', v'')$, then $ltl(\lambda, nodef) = (v' \wedge \mathbf{X}(v'' \wedge \mathbf{F}v_f))$.

By induction on the number of definitions in λ, it can be shown that a test sequence π exercises λ if and only if $\pi \models \mathbf{ltl}(\lambda)$. Figure 10 shows the ordered definition contexts, their LTL formulas, and the reduced subsumption graph for the flow graph in Figure 1.

5 Conclusions and Future Work

We have presented a method for reducing the cost of test generation for structural testing. We investigated a family of control flow and data flow oriented coverage criteria and reduced the problem of finding subsumption relations to the problem of LTL model checking. We illustrated the method using the flow graph model of a simple program module.

Our method can be applied to more accurate models of programs. Traditionally, test generation has been performed upon flow graphs. Since a flow graph preserves only the control flow and ignores the values of data variables, it is often the case that the size of state space is not a concern. However, test generation is increasingly performed upon more accurate models that respect the values of data variables such as reachability graphs and abstract state graphs obtained by abstract interpretation. In this case, the size of state space is the primary concern and model checking has been proven to be effective for controlling the state explosion problem. We plan to conduct case studies to see how large and complex programs can be handled by our method when reachability graphs or abstract state graphs are used.

Our method can also be applied to requirements specifications written in state-based specification languages such as extended finite state machines, statecharts, and SDL. Optimal test generation from such specifications is more complicated than that from program modules because it is necessary to cope with a rich set of language constructs for modeling hierarchy, concurrency, and communications. Our method is language-independent in that the temporal logic formulas employed in the method can be immediately used for various specification languages. In fact, differences between specification languages (for example, synchronous computational model in statecharts versus asynchronous computational model in SDL and communications through event broadcasting in statecharts versus communications through unbounded queues in SDL) only affect the rules for translating them into input to model checkers.

Acknowledgements

This research is supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada under grant OGP 976.

References

1. H. Agrawal, "Dominators, Super Blocks, and Program Coverage," *Proc. of the 21st ACM Symposium on Principles of Programming Languages*, pp. 25-34, 1994.
2. A.V. Aho, R. Sethi, and J.D. Ullman, *Compilers, Principles, Techniques, and Tools*, Addision-Wesley, 1986.
3. A. Bertolino, "Unconstrained Edges and Their Application to Branch Analysis and Testing of Programs," *The Journal of Systems and Software*, 20(2):125-133, Feb. 1993.
4. A. Bertolino and M. Marré, "Automatic Generation of Path Covers Based on the Control Flow Analysis of Computer Programs," *IEEE Transactions on Software Engineering*, 20(12):885-899, Dec. 1994.
5. A. Bertolino and M. Marré, "How Many Paths are Needed for Branch Testing?" *The Journal of Systems and Software*, 35(2):95-106, Nov. 1996.
6. T. Chusho, "Test Data Selection and Quality Estimation Based on the Concept of Essential Branches for Path Testing," *IEEE Transactions on Software Engineering*, 13(5):509-517, May 1987.
7. R. Dssouli, K. Saleh, E. Aboulhamid, A. En-Nouaary, and C. Bourhfir, "Test Development for Communication Protocols: towards Automation," *Computer Networks*, 31(7):1835-1872, June 1999.
8. R. Gupta and M.L. Soffa, "Employing Static Information in the Generation of Test Cases," *Software Testing, Verification and Reliability*, 3(1):29-48, 1993.
9. H.S. Hong, I. Lee, O. Sokolsky, and H. Ural, "A Temporal Logic Based Theory of Test Coverage and Generation," *TACAS '02*, Vol. 2280 of LNCS, pp. 327-341, Springer-Verlag, 2002.
10. H.S. Hong, S.D. Cha, I. Lee, O. Sokolsky, and H. Ural, "Data Flow Testing as Model Checking," *Proc. of the 25th International Conference on Software Engineering*, pp. 232-242, 2003.
11. J.W. Laski and B. Korel, "A Data Flow Oriented Program Testing Strategy," *IEEE Transactions on Software Engineering*, 9(5):347-354, May 1983.
12. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1992.
13. M. Marré and A. Bertolino, "Unconstrained Duas and Their Use in Achieving All-uses Coverage," *Proc. of the International Symposium on Software Testing and Analysis*, pp. 147-157, 1996.
14. M. Marré and A. Bertolino, "Reducing and Estimating the Cost of Test Coverage Criteria," *Proc. of the 18th International Conference on Software Engineering*, pp. 486-494, 1996.
15. S.C. Ntafos, "On Required Element Testing," *IEEE Transactions on Software Engineering*, 10(11):795-803, Nov. 1984.
16. S. Rapps and E.J. Weyuker, "Selecting Software Test Data Using Data Flow Information," *IEEE Transactions on Software Engineering*, 11(4):367-375, Apr. 1985.
17. S. Tasiran and K. Keutzer, "Coverage Metrics for Functional Validation of Hardware Designs," *IEEE Design and Test of Computers*, 18(4):36-45, July/Aug. 2001.
18. H. Ural and B. Yang, "A Test Sequence Generation Method for Protocol Testing," *IEEE Transactions on Communications*, 39(4):514-523, Apr. 1991.
19. H. Ural, K. Saleh, and A. Williams, "Test Generation Based on Control and Data Dependencies within System Specifications in SDL," *Computer Communications*, 23(7):609-627, Mar. 2000.
20. H. Zhu, P.A. Hall, and J.H.R. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, 29(4):366-427, Dec. 1997.