Dependence Testing: Extending Data Flow Testing with Control Dependence

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 new approach to structural testing, called dependence testing. First we propose dependence oriented coverage criteria that extend conventional data flow oriented coverage criteria with control dependence. This allows one to capture the full dependence information of a program or specification systematically. We then describe a model checking-based approach to test generation for dependence testing. It is shown that dependence oriented coverage criteria can be characterized in the temporal logics LTL and CTL. This enables one to use any LTL and CTL model checkers as test generators. Finally, we show that the temporal logic-based characterization can also be used for reducing the cost of dependence testing.

1 Introduction

In structural testing, we are given a coverage criterion defining a set of entities in the structure of a program or specification 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 two main types of structural testing. Control flow testing calls for exercising single entities such as statements, branches, decisions, and conditions. Data flow testing calls for exercising associations between definitions and uses of variables such as definition-use pairs and definition-use chains. These associations capture the dependence information of a program or specification mainly in terms of data dependence. Data flow testing has been widely used for program testing[34] and protocol conformance testing with formal specifications written in SDL and Estelle whose underlying model is extended finite state machine[11].

This paper presents a new approach to structural testing, called *dependence testing*. The main contributions of the paper are three-fold. First, we propose dependence oriented coverage criteria that extend conventional data flow oriented coverage criteria with control dependence. Our new coverage criteria are

F. Khendek and R. Dssouli (Eds.): TestCom 2005, LNCS 3502, pp. 23–39, 2005.
© IFIP 2005

motivated by the work of Podgurski and Clarke[30] which evaluates data flow oriented coverage criteria in terms of program dependence. In [30], it is shown that both data dependence and control dependence are necessary to detect the propagation of erroneous values caused by faults. It is also shown that although data flow oriented coverage criteria incorporate limited forms of control dependence, they are not powerful enough to detect the propagation of all of erroneous values. However, the question of how to extend data flow oriented coverage criteria has remained unanswered. In this paper, we show that the data flow oriented coverage criteria in [17, 28, 27, 32] can be naturally extended with control dependence. This allows one to capture the dependence information of a program or specification systematically based on both data dependence and control dependence.

Second, we discuss test generation for dependence testing. Recently there have been several proposals of model checking-based approaches to test generation for control flow testing[3, 7, 12, 14, 16, 29] and data flow testing[20, 21]. Model checking[9] is a formal verification technique for determining whether a system model satisfies a property written in temporal logic and model checkers such as SMV[26] and SPIN[18] are already used on a regular basis for the verification of real-world applications. In addition to being automatic, an important feature of model checking is the ability of explaining the success or failure of a temporal logic formula in terms of witnesses or counterexamples, respectively. The main idea of model checking-based test generation[3, 7, 12, 14, 16, 20, 21, 29] is to characterize test coverage in temporal logic in such a way that the problem of test generation is reduced to the problem of finding a set of witnesses or counterexamples for a set of temporal logic formulas. The capability of model checkers to construct witnesses and counterexamples enables efficient and scalable test generation. In this paper, we extend the model checking-based approach in [20, 21] for dependence testing. We show that dependence oriented coverage criteria can be characterized in the temporal logics LTL and CTL so that any LTL and CTL model checkers can be used as test generators for dependence testing.

Finally, we show that the temporal logic-based characterization of dependence oriented coverage criteria can also be used for reducing the cost of dependence testing. There have been several proposals of approaches to reducing the cost of control flow testing[1, 4, 5, 6, 8] and data flow testing[15, 24, 25]. The main idea of these approaches is to construct a subset of entities for a given 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 [24, 25], we call such a subset a spanning set for the coverage criteria. Recently in [22], the authors show that the problem of finding a minimum spanning set for a family of data flow oriented coverage criteria can be reduced to the model checking problem of LTL. In this paper, we extend the results of [22] and show how LTL model checking can be used for reducing the cost of dependence testing.

The remainder of the paper is organized as follows. After introducing preliminary definitions in Section 2, we investigate test coverage, generation, and reduction for dependence testing in Section 3, 4, 5, respectively. We conclude the paper with a discussion of future work in Section 6.

2 Preliminaries

This section recalls the basics of LTL and flow graph, which are the logic and model employed in our approach, respectively.

2.1 Logics: LTL and CTL

In this paper we will make use of both LTL and CTL. We give a brief introduction to LTL here and refer the interested readers to [9] for the syntax and semantics of CTL. A formula f in LTL is 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: $f := p \mid \neg f \mid f \wedge f \mid \mathbf{X}f \mid f\mathbf{U}f$ where $p \in AP$. We also use the temporal operators $\mathbf{F}$ (eventually) and $\mathbf{G}$ (always) defined by $\mathbf{F}f \equiv true\mathbf{U}f$ and $\mathbf{G}f \equiv \neg\mathbf{F}\neg f$.

The semantics of LTL is defined with respect to an infinite path $\pi = \sigma_0\sigma_1...$ where for every $i \geq 0$, σ_i is a subset of AP. For a position i, $\pi(i)$ is the i-th element of π and π^i is the suffix $\sigma_i\sigma_{i+1}...$ of π. We write $\pi \models f$ to indicate that π satisfies f.

- $\pi \models p$ iff $p \in \sigma_0$;
- $\pi \models \neg f$ iff $\pi \not\models f$;
- $\pi \models f_1 \wedge f_2$ iff $\pi \models f_1$ and $\pi \models f_2$;
- $\pi \models \mathbf{X}f$ iff $\pi^1 \models f$;
- $\pi \models f_1\mathbf{U}f_2$ iff there exists $i \geq 0$ such that $\pi^i \models f_2$ and $\pi^j \models f_1$ for every $0 \leq j < i$.

We also interpret LTL over a Kripke structure (Q, q_{init}, L, R) where Q is a set of states, $q_{init} \in Q$ is the initial state, $L : Q \rightarrow 2^{AP}$ labels each state with atomic propositions, and $R \subseteq Q \times Q$ is the total transition relation. We write $M \models f$ to indicate that for every infinite path π of M such that $\pi(0) = q_{init}$, $\pi \models f$. The model checking problem of LTL is to decide if for given M and f, it holds that $M \models f$.

2.2 Model: Flow Graph

Flow graphs are the standard model of programs in conventional program analysis and testing[2]. Flow graphs have also been used in analyzing and testing specification languages whose underlying model is extended finite state machine such as Estelle[32], SDL[33], and statecharts[19] as well as process algebra such as LOTOS[31].

A *flow graph* is a directed graph $G = (V, v_s, v_f, A)$ where V is a set of nodes, $v_s \in V$ is the start node, $v_f \in V$ is the final node, and $A \subseteq V \times V$ is a set of arcs. The start node v_s and final node v_f represent the single entry and single exit point, respectively. A node represents a simple statement (such as assignment, input, and output) or the predicate of a conditional or repetitive statement (such as if and while). An arc represents possible flow of control between statements. Each variable occurrence is classified as a definition or use. For a variable x and a node v, x is *defined* at v, denoted by $d(x, v)$, if x is assigned a value at v. x is *used* at v, denoted by $u(x, v)$, if v is referenced at v. A use $u(x, v)$ is a *computation-use* (c-use) if v represents a statement and is a *predicate-use* (p-use) if v represents a predicate. A path $v_1...v_n$ 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 simple program and its flow graph where v_1 is the start node and v_9 is the final node.

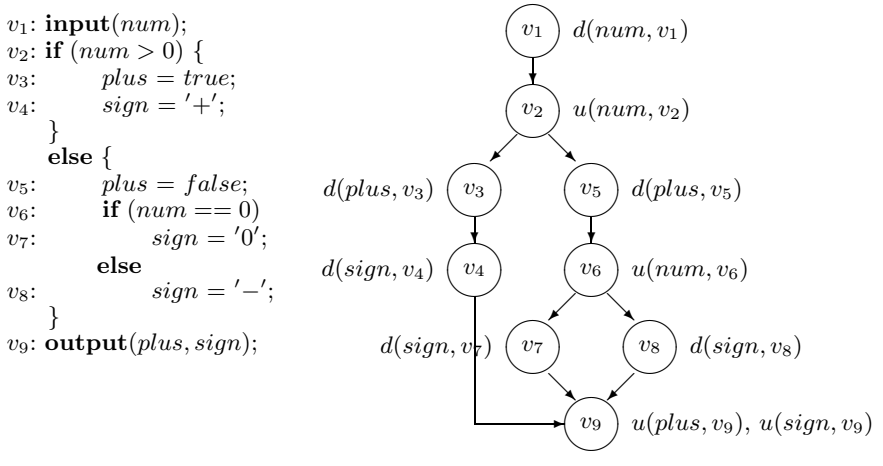


Fig. 1. A program and its flow graph

There are two types of program dependence. For two nodes v and v', we say that v *directly data-affects* v' *through variable* x (or equivalently, v' *is directly data-dependent on* v *through variable* x), denoted by $v \xrightarrow{x} v'$, if x is defined at v, x is used at v', and there is a path $vv_1...v_nv'$ such that x is not defined v_i for every $1 \leq i \leq n$. In this case, $v_1...v_n$ is a *definition-clear path* with respect to x. A test sequence *exercises* $v \xrightarrow{x} v'$ if $vv_1...v_nv'$ is a subpath of the test sequence where $v_1...v_n$ is a definition-clear path with respect to x. We say that v' *postdominates* v if every path from v to v_f contains v' and that v *directly control-affects* v' (or equivalently, v' is *directly control-dependent* on v), denoted by $v \xrightarrow{c} v'$, if v has two successors v_1 and v_2 such that v' postdominates v_1 but v' does not postdominate v_2. A test sequence *exercises* $v \xrightarrow{c} v'$ if $vv_1...v_nv'$ is a subpath of the test sequence.

3 Test Coverage

Let $v_1, v_2, ..., v_n$ be nodes. We say that $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_n$ is a *dependence-chain* if for every $1 \leq i < n$, $v_i \rightarrow v_{i+1}$ is either a direct data dependence $v_i \xrightarrow{x} v_{i+1}$ or direct control dependence $v_i \xrightarrow{c} v_{i+1}$. Obviously the strongest coverage criterion based on the dependence information, which we call *all-dependence-chains coverage criterion*, is to require that every dependence-chain be exercised. However, this is in general impossible to achieve since the number of dependence-chains in a program may be large or even infinite in the presence of loops. In this section we investigate the data flow oriented coverage criteria in [17, 28, 27, 32], which capture the dependence information mainly in terms of data dependence, and extend them with control dependence. This allows one to generate test suites consisting of a finite and reasonable number of test sequences based on both data dependence and control dependence.

3.1 Direct Dependences

All-Dependence-Pairs Coverage Criterion. A pair $(d(x, v), u(x, v'))$ is a *definition-use pair* (du-pair) if there is a path $vv_1...v_nv'$ such that $v_1...v_n$ is a definition-clear path with respect to x. A test suite Π satisfies *reach coverage criterion*[17] if every du-pair $(d(x, v), u(x, v'))$ is exercised by some test sequence in Π.

It is straightforward to rephrase reach coverage criterion in terms of program dependence: A test suite Π satisfies *reach coverage criterion* if every direct data dependence $v \xrightarrow{x} v'$ is exercised by some test sequence in Π.

We extend reach coverage criterion with direct control dependence as follows: A test suite Π satisfies *all-dependence-pairs coverage criterion* if Π satisfies reach coverage criterion and every direct control dependence $v \xrightarrow{c} v'$ is exercised by some test sequence in Π. In Figure 1, all-dependence-pairs coverage criterion requires that the following direct data and control dependences be exercised.

- direct data dependences: $v_1 \xrightarrow{num} v_2$, $v_1 \xrightarrow{num} v_6$, $v_3 \xrightarrow{plus} v_9$, $v_5 \xrightarrow{plus} v_9$, $v_4 \xrightarrow{sign} v_9$, $v_7 \xrightarrow{sign} v_9$, $v_8 \xrightarrow{sign} v_9$
- direct control dependences: $v_2 \xrightarrow{c} v_3$, $v_2 \xrightarrow{c} v_4$, $v_2 \xrightarrow{c} v_5$, $v_2 \xrightarrow{c} v_6$, $v_6 \xrightarrow{c} v_7$, $v_6 \xrightarrow{c} v_8$

All-Dependence-Pairs-with-Puses Coverage Criterion. Rapps and Weyuker's criteria[28] extend reach coverage criterion by distinguishing between c-uses and p-uses. A du-pair $(d(x, v), u(x, v'))$ is a *definition-cuse pair* (dcu-pair) if $u(x, v')$ is a c-use. Otherwise, it is a *definition-puse pair* (dpu-pair). Let $(d(x, v), u(x, v'))$ be a dpu-pair and v'' be a successor of v'. A test sequence *exercises* $(d(x, v), u(x, v'), v'')$ if $vv_1...v_nv'v''$ is a subpath of the test sequence where $v_1...v_n$ is a definition-clear path with respect to x. A test suite Π satisfies *all-uses coverage criterion*[28] if for every dcu-pair $(d(x, v), u(x, v'))$, the dcu-pair is exercised by some test sequence in Π and for every dpu-pair $(d(x, v), u(x, v'))$ and every successor v'' of v', $(d(x, v), u(x, v'), v'')$ is exercised by some test sequence in Π.

In all-uses coverage criterion, a test sequence exercising $(d(x,v),u(x,v'))$ reflects the direct data dependence $v \xrightarrow{x} v'$, whereas a test sequence exercising $(d(x,v),u(x,v'),v'')$ reflects the dependence-chain $v \xrightarrow{x} v' \xrightarrow{c} v''$, that is, the direct data dependence $v \xrightarrow{x} v'$ and direct control dependence $v' \xrightarrow{c} v''$ at the same time.

We extend all-uses coverage criterion with direct control dependence as follows: A test suite Π satisfies *all-dependence-pairs-with-puses coverage criterion* if Π satisfies all-uses coverage criterion and every direct control dependence $v \xrightarrow{c} v'$ is exercised by some test sequence in Π. In Figure 1, all-dependence-pairs-with-puses coverage criterion requires that the following dependences be exercised.

- dcu-pairs: $v_3 \xrightarrow{plus} v_9$, $v_5 \xrightarrow{plus} v_9$, $v_4 \xrightarrow{sign} v_9$, $v_7 \xrightarrow{sign} v_9$, $v_8 \xrightarrow{sign} v_9$
- dpu-pairs: $v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_3$, $v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_5$, $v_1 \xrightarrow{num} v_6 \xrightarrow{c} v_7$, $v_1 \xrightarrow{num} v_6 \xrightarrow{c} v_8$
- direct control dependences: $v_2 \xrightarrow{c} v_3$, $v_2 \xrightarrow{c} v_4$, $v_2 \xrightarrow{c} v_5$, $v_2 \xrightarrow{c} v_6$, $v_6 \xrightarrow{c} v_7$, $v_6 \xrightarrow{c} v_8$

3.2 Indirect Dependences

All-k-Dependence-Chains Coverage Criterion. While reach coverage criterion and all-uses coverage criterion focus on definitions and uses of the same variable, Ntafos' criteria[27] emphasize interactions among different variables. These interactions are captured in terms of sequences of du-pairs. A sequence $[(d(x_1,v_1),u(x_1,v_1'))\ ...\ (d(x_n,v_n),u(x_n,v_n'))]$ of du-pairs is a *data flow chain* (df-chain)[32] if for every $1 \leq i < n$, $v_i' = v_{i+1}$, that is, $u(x_i,v_i')$ and $d(x_{i+1},v_{i+1})$ occur at the same node and hence x_{i+1} is defined in terms of x_i. A path $v_1\pi_1v_2...v_n\pi_nv_n'$ is an *interaction path* of a df-chain if for every $1 \leq i \leq n$, π_i is a definition-clear path with respect to x_i. A test sequence *exercises* a df-chain if an interaction path of the df-chain is a subpath of the test sequence. A test suite Π satisfies *required k-tuples coverage criterion* if every df-chain consisting of k' du-pairs, $1 \leq k' < k$, is exercised by some test sequence in Π.

We rephrase required k-tuples coverage criterion in terms of program dependence: A test suite Π satisfies required k-tuples coverage criterion if every dependence-chain consisting of k' direct data dependences, $1 \leq k' < k$, is exercised by some test sequence in Π. Since required k-tuples coverage criterion is based on data dependence, it can only partially capture the dependence information. For example, consider the nodes v_1 and v_3 in Figure 1. Although there is a dependence-chain $v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_3$ from v_1 to v_3, required k-tuples coverage criterion fails to capture this dependence-chain since it contains control dependence.

We extend required k-tuples coverage criterion with control dependence as follows: A test suite Π satisfies *all-k-dependence-chains coverage criterion* if every dependence-chain consisting of k' direct dependences, $1 \leq k' < k$, is exercised by some test sequence in Π. We note that required 2-tuples coverage criterion (resp. all-2-dependence-chains coverage criterion) is equivalent to reach coverage criterion (resp. all-dependence-pairs coverage criterion). In Figure 1, all-

3-dependence-chains coverage criterion requires that the following dependence-chains be exercised[1].

$$v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_3,\ v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_4,\ v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_5,\ v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_6,$$
$$v_1 \xrightarrow{num} v_6 \xrightarrow{c} v_7,\ v_1 \xrightarrow{num} v_6 \xrightarrow{c} v_8,$$
$$v_2 \xrightarrow{c} v_3 \xrightarrow{plus} v_9,\ v_2 \xrightarrow{c} v_4 \xrightarrow{sign} v_9,\ v_2 \xrightarrow{c} v_5 \xrightarrow{plus} v_9,\ v_2 \xrightarrow{c} v_6 \xrightarrow{c} v_7,$$
$$v_2 \xrightarrow{c} v_6 \xrightarrow{c} v_8,$$
$$v_6 \xrightarrow{c} v_7 \xrightarrow{sign} v_9,\ v_6 \xrightarrow{c} v_8 \xrightarrow{sign} v_9$$

Let $[(d(x_1, v_1), u(x_1, v_1')) \ldots (d(x_n, v_n), u(x_n, v_n'))]$ be a df-chain. We have that $u(x_i, v_i')$ is a c-use for every $1 \leq i < n$ and the last use $u(x_n, v_n')$ may be either a c-use or p-use. By distinguishing between c-uses and p-uses, all-k-dependence-chains-with-puses coverage criterion may be defined. Due to space limit this coverage criterion will not be pursued in this paper.

All-IO-Dependence-Chains Coverage Criterion. Ural *et al.*'s coverage criteria[32, 33] also emphasize interactions among 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 [32, 33] considers df-chains 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. For example, in Figure 1, there are one input $d(num, v_1)$ and two outputs $u(plus, v_9)$ and $u(sign, v_9)$. 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. Since the number of df-chains from an input to an output may be infinite, we consider only *simple* df-chains that are allowed to have at most one occurrence of each du-pair. A test suite Π satisfies *all-IO-df-chains coverage criterion* if for every input i, every output o, and every simple df-chain from i to o, the df-chain is exercised by some test sequence in Π.

As is done by required k-tuples coverage criterion, all-IO-df-chains coverage criterion also partially captures the dependence information in terms of only data dependence. For example, consider the input $d(v_1, num)$ and output $u(v_9, plus)$ in Figure 1. There are several dependence chains from v_1 to v_9, say $v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_3 \xrightarrow{plus} v_9$, but all-IO-df-chains coverage criterion fails to capture those dependence-chains since they contain control dependence.

We extend all-IO-df-chains coverage criterion with control dependence as follows: A test suite Π satisfies *all-IO-dependence-chains coverage criterion* if for every input i, every output o, and every simple dependence-chain from i to o, the dependence-chain is exercised by some test sequence in Π. In Figure 1, all-IO-dependence-chains coverage criterion requires that the following dependence-chains be exercised.

$$v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_3 \xrightarrow{plus} v_9,\ v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_4 \xrightarrow{sign} v_9,\ v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_5 \xrightarrow{plus} v_9,$$

[1] Dependence-chains consisting of one direct dependence are not shown.

$v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_6 \xrightarrow{c} v_7 \xrightarrow{sign} v_9$, $v_1 \xrightarrow{num} v_2 \xrightarrow{c} v_6 \xrightarrow{c} v_8 \xrightarrow{sign} v_9$,
$v_1 \xrightarrow{num} v_6 \xrightarrow{c} v_7 \xrightarrow{sign} v_9$, $v_1 \xrightarrow{num} v_6 \xrightarrow{c} v_8 \xrightarrow{sign} v_9$

3.3 The Relationships Among Coverage Criteria

For two coverage criteria C_1 and C_2, C_1 *subsumes* C_2 if every test suite satisfying C_1 also satisfies C_2[28]. By definition, the four data flow oriented coverage criteria considered in this section are subsumed by their dependence oriented counterparts. For the other direction, the data flow oriented coverage criteria except all-uses coverage criterion do not subsume their counterparts.

It is interesting to investigate the relationship between all-uses coverage criterion and all-dependence-pairs-with-puses coverage criterion. Let A_P be the set of arcs starting from a node representing a predicate and A_C be the set of arcs (v, v') such that v directly control-affects v'. For example, in Figure 1,

$A_P = \{(v_2, v_3), (v_2, v_5), (v_6, v_7), (v_6, v_8)\}$ and
$A_C = \{(v_2, v_3), (v_2, v_4), (v_2, v_5), (v_2, v_6), (v_6, v_7), (v_6, v_8)\}$.

It is not hard to see that a test suite exercises every arc in A_P if and only if the test suite exercises every arc in A_C. Hence it follows that all-uses coverage criterion and all-dependence-pairs-with-puses coverage criterion subsume each other and hence they are equivalent with respect to the subsume relation.

One of the limitations of the subsume relation is that it does not always guarantee a better fault-detecting ability, that is, C_1 subsumes C_2 but there are test suites that satisfy C_2 that expose faults while test suites that satisfy C_1 do not expose any faults. The cover and properly cover relations in [13] address this limitation. For a coverage criterion C, let $SD(C)$ be the multiset of subdomains such that C requires the selection of one or more input values from each subdomain in $SD(C)$. C_1 *covers* C_2 if for every subdomain $D \in SD(C_2)$, there is a collection $\{D_1, ..., D_n\}$ in $SD(C_1)$ such that $D_1 \cup ... \cup D_n = D$. Roughly speaking, C_1 *properly covers* C_2 if C_1 covers C_2 and, in addition, the number of times a subdomain D_1 in $SD(C_1)$ is used to characterize the subdomains in $SD(C_2)$ is at most the number of times D_1 appears in $SD(C_1)$. It is not hard to see that if $SD(C_1)$ is a superset of $SD(C_2)$, then C_1 covers C_2 and C_1 properly covers C_2. We have that the multisets of subdomains for the dependence oriented coverage criteria defined in this section are supersets of those for their data flow oriented counterparts. Hence the dependence oriented coverage criteria cover and properly cover their data flow oriented counterparts but not vice versa.

4 Test Generation

This section shows how test generation for dependence testing can be formulated in the temporal logics LTL and CTL. We restrict ourselves to a fragment of LTL consisting of guarantee formulas. An LTL formula is a *guarantee formula* if there is a finite path π such that for every infinite path π', $\pi \cdot \pi'$ satisfies the formula.

The finite prefix π is called a *witness* of the formula. Intuitively, it is sufficient to use a finite path to explain the success of a guarantee formula while we need an infinite path for a general LTL formula. For a set F of guarantee formulas and a set Π of finite paths, we say that Π is a *witness-set* of F if for every formula f in F, there is a finite path in Π that is a witness of f. In the following sections, we show that test generation for dependence oriented coverage criteria can be reduced to the problem of finding a witness-set of guarantee formulas.

4.1 Direct Dependences

All-Dependence-Pairs Coverage Criterion. Let $def(x)$ be the disjunction of nodes at which x is defined. For example, in Figure 1, $def(num) ::= v_1$, $def(plus) ::= v_3 \vee v_5$, and $def(sign) ::= v_4 \vee v_7 \vee v_8$. For a direct data dependence $v \xrightarrow{x} v'$, we associate an LTL formula defined by

$$\mathbf{ltl}(v \xrightarrow{x} v') = \mathbf{F}(v \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{F}v_f)])$$

with the property that a finite path π is a test sequence exercising $v \xrightarrow{x} v'$ 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 π is a witness of $\mathbf{ltl}(v \xrightarrow{x} v')$. For example, consider the direct data dependence $v_1 \xrightarrow{num} v_6$ in Figure 1. A test sequence exercising $v_1 \xrightarrow{num} v_6$ is shown in Figure 2, which is also a witness of $\mathbf{F}(v_1 \wedge \mathbf{X}[\neg def(num)\mathbf{U}(v_6 \wedge \mathbf{F}v_9)])$.

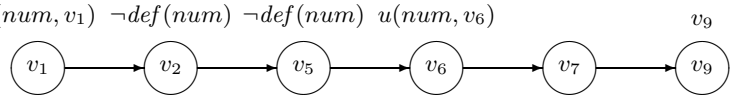


Fig. 2. A test sequence exercising $v_1 \xrightarrow{num} v_6$

For a direct control dependence $v \xrightarrow{c} v'$, we associate an LTL formula defined by

$$\mathbf{ltl}(v \xrightarrow{c} v') = \mathbf{F}(v \wedge \mathbf{XF}(v' \wedge \mathbf{F}v_f))$$

with the property that a finite path π is a test sequence exercising $v \xrightarrow{c} v'$ if and only if there are $0 \leq i < j \leq k$ such that $\pi(i) \models v$, $\pi(j) \models v'$, and $\pi(k) \models v_f$ if and only if π is a witness of the LTL formula $\mathbf{ltl}(v \xrightarrow{c} v')$. For example, consider the direct control dependence $v_2 \xrightarrow{c} v_6$ in Figure 1. A test sequence exercising $v_2 \xrightarrow{c} v_6$ is shown in Figure 3, which is also a witness of $\mathbf{F}(v_2 \wedge \mathbf{XF}(v_6 \wedge \mathbf{F}v_9))$.

We characterize all-dependence-pairs coverage criterion in terms of witness-sets. A test suite Π satisfies all-dependence-pairs coverage criterion if and only if Π is a witness-set of

$$\bigcup_{v \xrightarrow{x} v'} \mathbf{ltl}(v \xrightarrow{x} v') \cup \bigcup_{v \xrightarrow{c} v'} \mathbf{ltl}(v \xrightarrow{c} v').$$

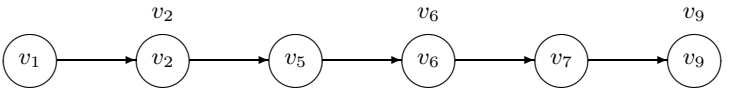


Fig. 3. A test sequence exercising $v_2 \xrightarrow{c} v_6$

CTL can also be used in the characterization of dependence oriented coverage criteria. A finite path is a witness of $\mathbf{ltl}(v \xrightarrow{x} v')$ if and only if the finite path is a witness of the CTL formula defined by

$$\mathbf{ctl}(v \xrightarrow{x} v') = \mathbf{EF}(v \wedge \mathbf{EX}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{EF}v_f)]).$$

A finite path is a witness of $\mathbf{ltl}(v \xrightarrow{c} v')$ if and only if the finite path is a witness of the CTL formula defined by

$$\mathbf{ctl}(v \xrightarrow{c} v') = \mathbf{EF}(v \wedge \mathbf{EXEF}(v' \wedge \mathbf{EF}v_f)).$$

All-Dependence-Pairs-with-Puses Coverage Criterion. For a dependence chain $v \xrightarrow{x} v' \xrightarrow{c} v''$, we associate an LTL formula defined by

$$\mathbf{ltl}(v \xrightarrow{x} v' \xrightarrow{c} v'') = \mathbf{F}(v \wedge \mathbf{X}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{XF}(v'' \wedge \mathbf{F}v_f))])$$

and a CTL formula defined by

$$\mathbf{ctl}(v \xrightarrow{x} v' \xrightarrow{c} v'') = \mathbf{EF}(v \wedge \mathbf{EX}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{EXEF}(v'' \wedge \mathbf{EF}v_f))]).$$

A test suite Π satisfies all-dependence-pairs-with-puses coverage criterion if and only if Π is a witness-set of

$$\bigcup_{v \xrightarrow{x} v'} \mathbf{ltl}(v \xrightarrow{x} v') \cup \bigcup_{v \xrightarrow{x} v' \xrightarrow{c} v''} \mathbf{ltl}(v \xrightarrow{x} v' \xrightarrow{c} v'') \cup \bigcup_{v \xrightarrow{c} v'} \mathbf{ltl}(v \xrightarrow{c} v').$$

4.2 Indirect Dependences

All-k-Dependence-Chains Coverage Criterion. For a dependence-chain κ, we associate an LTL formula defined by

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

The CTL formula $\mathbf{ctl}(\kappa)$ is defined in a similar way.

- $\mathbf{ctl}(\kappa) = \mathbf{EF}ctl(\kappa)$,
- if κ is $v \xrightarrow{x} v'$, then $ctl(\kappa) = (v \wedge \mathbf{EX}[\neg def(x)\mathbf{U}(v' \wedge \mathbf{EF}v_f)])$,
- if κ is $v \xrightarrow{c} v'$, then $ctl(\kappa) = (v \wedge \mathbf{EXEF}(v' \wedge \mathbf{EF}v_f))$,

– if κ is $v \xrightarrow{x} v' \cdot \kappa'$, then $ctl(\kappa) = (v \wedge \mathbf{EX}[\neg def(x)\mathbf{U}\, ctl(\kappa')])$,
– if κ is $v \xrightarrow{c} v' \cdot \kappa'$, then $ctl(\kappa) = (v \wedge \mathbf{EXEF}(v' \wedge \mathbf{EF}\, ctl(\kappa')))$.

By induction on the number of du-pairs in κ, it can be shown that a finite path is a test sequence exercising a df-chain κ if and only if the finite path is a witness of $\mathbf{ltl}(\kappa)$ if and only if the finite path is a witness of $\mathbf{ctl}(\kappa)$.

A test suite Π satisfies all-k-dependence-chains coverage criterion if and only if Π is a witness-set of

$$\bigcup_{\kappa \in DC(1) \cup \ldots \cup DC(k-1)} \mathbf{ltl}(\kappa)$$

where $DC(n)$ is a set of dependence-chains consisting of n direct dependences.

All-IO-Dependence-Chains Coverage Criterion. A test suite Π satisfies all-IO-dependence-chains coverage criterion if Π is a witness-set of

$$\bigcup_{i} \bigcup_{o} \bigcup_{\kappa \in SDC(i,o)} \mathbf{ltl}(\kappa)$$

where $SDC(i, o)$ is a set of simple dependence-chains from input i to output o.

5 Test Reduction

This section shows how the problem of test reduction for dependence testing can be formulated as the LTL model checking problem.

5.1 Subsumption Graph

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 $E(G, C)$. Hence a test suite exercises every entity in a spanning set if and only if the test suite satisfies the coverage criterion. A *minimum spanning set* is a spanning set S such that $|S| \leq |S'|$ for every spanning set S'. 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.

We construct a minimum spanning set using *subsumption graph* and *reduced subsumption graph*. 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 node. Let $v_1, ..., v_n$ be the nodes of the reduced subsumption graph that have no incoming arcs, that is, the nodes that are not subsumed by any other nodes. 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 $1 \leq i \leq n$.

Figure 4 shows an algorithm for finding a subsumption graph in a generic fashion without being specific about any coverage criteria. For every pair (e, e') of entities, 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. The correctness of the algorithm can be understood as follows. Let $e, e' \in E(G, C)$. e subsumes e' if and only if for every finite path π, π is a test sequence exercising e implies π is a test sequence exercising e' if and only if for every finite path π, π is a witness of $\mathbf{ltl}(e)$ implies π is a witness of $\mathbf{ltl}(e')$ if and only if for every finite path π, π is a witness of $\mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ if and only if for every infinite path π, $\pi \models \mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ if and only if $G \models \mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$.

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

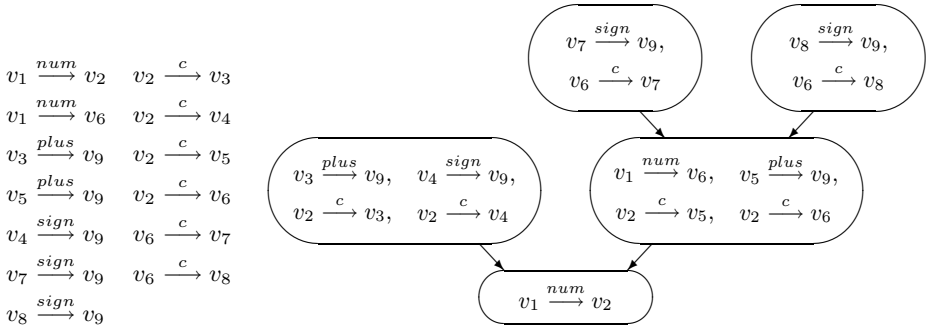
1: construct the set $E(G, C)$ of entities of G required by C;
2: $SR := \emptyset$;
3: **for** every pair (e, e'), $e, e' \in E(G, C)$, $e \neq e$ **do**
4: model check $\mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ against G;
5: **if** $G \models \mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ **then**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

Fig. 4. An algorithm for constructing a subsumption graph

For example, consider all-dependence-pairs coverage criterion in Figure 1. The set of entities required to be covered are shown in Figure 5.(a). By model-checking the formula $\mathbf{ltl}(e) \rightarrow \mathbf{ltl}(e')$ for every pair (e, e') of entities, we construct the subsumption graph. We then construct the reduced subsumption graph by collapsing each strongly connected component in the subsumption graph into one node. Figure 5.(b) shows the reduced subsumption graph. Finally we construct a minimum spanning set by selecting one entity from each of the strongly connected components $\{v_3 \xrightarrow{plus} v_9, v_4 \xrightarrow{sign} v_9, v_2 \xrightarrow{c} v_3, v_2 \xrightarrow{c} v_4\}$, $\{v_7 \xrightarrow{sign} v_9, v_6 \xrightarrow{c} v_7\}$, and $\{v_8 \xrightarrow{sign} v_9, v_6 \xrightarrow{c} v_8\}$ that have no incoming arcs.

5.2 Subsumption Forest

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 is used to identify all possible minimum spanning sets. If we are only



(a) $E(G, C)$ (b) reduced subsumption graph

Fig. 5. The reduced subsumption graph for Figure 1 and all-dependence-pairs coverage criterion

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 6. The intuition behind the algorithm is that if e_i subsumes e_j (Line 10) then we do not consider e_j any more between Lines 5 and 12, which reduces the number of model checking that needs to be performed. 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.

Figure 7 shows a subsumption forest for Figure 1 and all-dependence-pairs coverage criterion. We construct a minimum spanning set by finding the root nodes of the subsumption forest: $\{v_3 \xrightarrow{plus} v_9, v_7 \xrightarrow{sign} v_9, v_8 \xrightarrow{sign} v_9\}$.

INPUT: a flow graph G and a coverage criterion C
OUTPUT: a spanning forest $(E(G, C), SF)$

```
1: let E(G,C) be {e_1, ..., e_n};
2: SF := ∅;
3: for i := 1 to n do
4:     marked[i] := false;
5: for i := 1 to n do
6:     if marked[i] = false then
7:         for j := 1 to n, j ≠ i do
8:             if marked[j] = false then
9:                 model check ltl(e_i) → ltl(e_j) against G;
10:                if G ⊨ ltl(e_i) → ltl(e_j) then /* e_i subsumes e_j */
11:                    SF := SF ∪ {(e_i, e_j)};
12:                    marked[j] := true;
13: return (E(G,C), SF);
```

Fig. 6. An algorithm for constructing a spanning forest

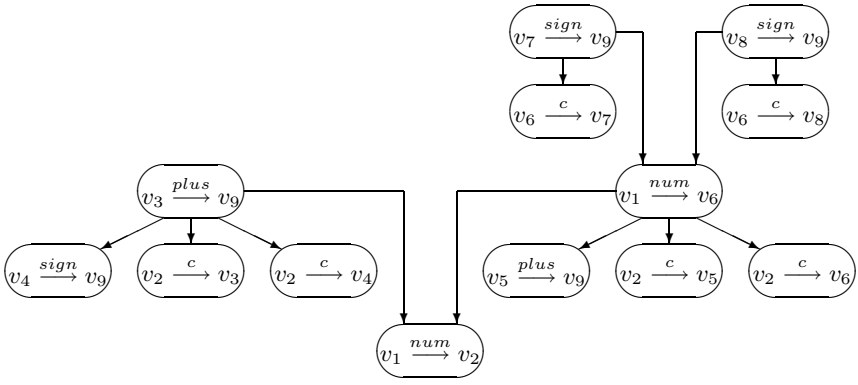


Fig. 7. The subsumption forest for Figure 1 and all-dependence-pairs coverage criterion

6 Conclusions and Future Work

We have presented an approach to structural testing, called dependence testing. For test coverage, we have extended data flow oriented coverage criteria with control dependence in order to capture the dependence information of a program or specification in terms of both data dependence and control dependence. For test generation, we have showed that dependence oriented coverage criteria can be characterized in temporal logic in such a way that test generation can be reduced to the problem of finding witnesses for LTL or CTL formulas. For test reduction, we have showed that the LTL-based characterization can also be used for reducing the cost of dependence testing. It will be interesting to empirically study the extent to which dependence testing actually provides tests which are more effective at identifying errors, provides better reliability for programs under test, or exhibits a better cost ratio for test development.

Our approach 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 approach when reachability graphs or abstract state graphs are used.

Our approach can also be applied to requirements specifications written in state-based specification languages such as extended finite state machines, statecharts, and SDL. Test generation for such specifications is very different from that for programs since the specification languages typically provide a rich set

of language constructs for modeling hierarchy, concurrency, and communications. Our approach is language-independent in the sense that the temporal logic formulas employed in the approach can be immediately used for various specification languages. In fact, the differences among 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 message queues in SDL) only affect the rules for translating specifications into input to model checkers.

Acknowledgments

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

References

1. H. Agrawal, "Dominators, Super Blocks, and Program Coverage," *Proceedings 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. P. Ammann, P. Black, and W. Majurski, "Using Model Checking to Generate Tests from Specifications," in *Proceedings of the 2nd IEEE International Conference on Formal Engineering Methods*, pp. 46-54, 1998.
4. 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.
5. 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.
6. 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.
7. D. Beyer, A.J. Chlipala, T.A. Henzinger, R. Jhala, and R. Majumdar, "Generating Tests from Counterexamples," *Proceedings of the 26th International Conference on Software Engineering*, pp. 326-335, 2004.
8. 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.
9. E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking*, The MIT Press, 1999.
10. L.A. Clarke, A. Podgurski, D.J. Richardson, and S.J. Zeil, "A Formal Evaluation of Data Flow Path Selection Criteria," *IEEE Transactions on Software Engineering*, 15(11):1318-1332, Nov. 1989.
11. 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.

12. A. Engels, L. Feijs, and S. Mauw, "Test Generation for Intelligent Networks Using Model Checking," in *TACAS '97*, Vol. 1217 of LNCS, pp. 384-398, Springer-Verlag, 1997.
13. P.G. Frankl and E.J. Weyuker, "A Formal Analysis of the Fault-Detecting Ability of Testing Methods," *IEEE Transactions on Software Engineering*, 19(3):202-213, Mar. 1993.
14. A. Gargantini and C. Heitmeyer, "Using Model Checking to Generate Tests from Requirements Specifications," in *Proceedings of ESEC/FSE '99* pp. 146-162, 1999.
15. 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.
16. M.P. Heimdahl, S. Rayadurgam, W. Visser, G. Devaraj, and J. Gao, "Auto-Generating Test Sequences Using Model Checkers: A Case Study," *Proceedings of the 3th International Workshop on Formal Approaches to Testing of Software*, Vol. 2931 of LNCS, pp. 44-62, Springer, 2003.
17. P.M. Herman, "A Data Flow Approach to Program Testing," *Australian Computer Journal*, 8(3):92-96, Nov. 1976.
18. G.J. Holzmann, "The Model Checker SPIN," *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, pp. 279-295, May 1997.
19. H.S. Hong, Y.G. Kim, S.D. Cha, D.H. Bae, and H. Ural, "A Test Sequence Selection Method for Statecharts," *Journal of Software Testing, Verification, and Reliability*, 10(4):203-227, Dec. 2000.
20. 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, 2002.
21. H.S. Hong, S.D. Cha, I. Lee, O. Sokolsky, and H. Ural, "Data Flow Testing as Model Checking," *Proceedings of the 25th International Conference on Software Engineering*, pp. 232-242, 2003.
22. H.S. Hong and H. Ural, "Using Model Checking for Reducing the Cost of Test Generation," *Proceedings of the 4th International Workshop on Formal Approaches to Testing of Software*, LNCS, Springer, 2004.
23. J.W. Laski and B. Korel, "A Data Flow Oriented Program Testing Strategy," *IEEE Transactions on Software Engineering*, 9(5):347-354, May 1983.
24. M. Marré and A. Bertolino, "Unconstrained Duas and Their Use in Achieving All-uses Coverage," *Proceedings of the International Symposium on Software Testing and Analysis*, pp. 147-157, 1996.
25. M. Marré and A. Bertolino, "Reducing and Estimating the Cost of Test Coverage Criteria," *Proceedings of the 18th International Conference on Software Engineering*, pp. 486-494, 1996.
26. K.L. McMillan, *Symbolic Model Checking − an Approach to the State Explosion Problem*, Kluwer Academic Publishers, 1993.
27. S.C. Ntafos, "On Required Element Testing," *IEEE Transactions on Software Engineering*, 10(11):795-803, Nov. 1984.
28. 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.
29. S. Rayadurgam and M.P. Heimdahl, "Coverage Based Test Generation Using Model Checkers," *Proceedings of the 8th Annual IEEE International Conference on the Engineering of Computer Based Systems*, pp. 83-91, 2001.
30. A. Podgurski and L.A. Clarke, "A Formal Model of Program Dependences and Its Implications for Software Testing, Debugging, and Maintenance," *IEEE Transactions on Software Engineering*, 16(9):965-979, Sept. 1990.

31. H. van der Schoot and H. Ural, "Data Flow Oriented Test selection for LOTOS," *Computer Networks*, 27(7):1111-1136, 1995.
32. H. Ural and B. Yang, "A Test Sequence Generation Method for Protocol Testing," *IEEE Transactions on Communications*, 39(4):514-523, Apr. 1991.
33. 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.
34. 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.