Automatic Generation of Test Vectors for SCR-Style Specifications

Mark R. Blackburn
Software Productivity Consortium
blackbur@software.org

Robert D. Busser
T-VEC Technologies
bobb@ns.upcyber.com

Joseph S. Fontaine
Software Productivity Consortium
fontaine@software.org

Abstract

This paper provides the basis for integrating the Software Cost Reduction (SCR) specification method with the T-VEC (Test VECtor) test vector generator and specification analysis system. The SCR model is mapped to the T-VEC model to support automatic test vector generation for SCR specifications. The T-VEC system generated test vectors for an example SCR specification that was translated into the T-VEC language. The relationships between the models and the resulting test vectors are described. Two general guidelines for the translation process were identified that are fundamental for testing specifications that use event operators and for structuring the specifications to provide tests for all specified requirements.

1. Introduction

The benefits and cost savings of identifying problems during the requirements development phase are well known, and methods and tools have matured to make this process feasible in industry [HLK95, BB96]. For high assurance systems, formal approaches are being applied in industry. However, based on a survey of 12 industrial applications of formal methods, there were no tools used to automatically generate tests from the specifications that were developed and analyzed in support of the system verifications [CGR93]. Testing can account for 40% to 70% of the development effort [Bei83; GW94]. Testing a critical system can require tens or hundreds of thousands of test cases.[1] Such tools are valuable in reducing manual effort and preventing manual errors in the testing process, while freeing developers to focus on the more complex task of specification development and analysis. The combination of requirement and design specification methods and tools with automatic test generation technologies could significantly reduce the cost of verification and testing. Such integration would provide stronger arguments and benefits to use and further advance specification-based methods and tools.

SCR was one of the methods surveyed in the survey of formal methods industrial applications. It was developed at the Naval Research Laboratory (NRL) [Hen80]. The SCR method has a long history, and other methods like the Consortium Requirements Engineering Method (CoRE) [SPC93; FBWK92], have evolved from the original SCR method. Enhancements to CoRE have also been factored into the current embodiment of the SCR method. More recently, NRL has built tools to support specification acquisition, simulation, and formal analysis [HLK95; HBGL95; HJL96]. Powerful proof systems like PVS have been used to check well-formedness properties in SCR specifications [ORS95]. Model checking tools have also been applied to SCR-style specifications [ORS95; SA96]. These methods and tools are important in helping to develop correct specifications. However, there is still a need to assess that an operational system complies with its specification. Testing can be used in this type of assessment.

T-VEC is an integrated development environment and associated specification and verification method [BB96]. It was used to develop two avionics systems that were certified by the Federal Aviation Administration (FAA) based on DO-178A - *Software Considerations in Airborne Systems and Equipment Certification* [RTCA92] (now DO-178B). These certification guidelines emphasize a software engineering approach, where requirement-based testing and analysis are key to supporting the assurance arguments required for certification.

One of the key tools of the T-VEC system is an automatic ***test vector generator***; it determines test inputs, expected outputs, and a mapping of each test to the associated requirement, directly from formal specifications. A test vector generator that determines

expected output values can reduce the testing effort as compared to a test case generator, where the expected output values must be determined manually. The other tools of the environment check that the specification is well-formed with respect to the model, and the specification-based coverage analyzer ensures that every unique requirement specification has at least one corresponding test vector.

1.1 Overview

This paper describes the results of an effort to integrate the SCR formal methods approach with the T-VEC automatic test vector generator and specification analysis system. An example SCR specification taken from Heitmeyer et al. [HLK95; HJL96] was translated into the T-VEC specification language. Test vectors were generated and analyzed for several different variations of the translated specification. Although the models are very similar, the analysis of the resulting test vectors helped identify two general guidelines for the translation process:

- Variables referenced in an SCR event operator must be expanded into two states when translated into the T-VEC model to adequately represent the state before and after the event. T-VEC must generate input values for each variable at both states for all specified mode combinations of the SCR variables.
- The structure of the T-VEC specification must map to the ideal functions of an SCR specification so that only those constraints directly associated with an ideal function are relevant to the test vector generation and coverage analysis process.

These guidelines are fundamental for generating tests from specifications that use event operators to ensure that the reactive aspects of the system are implemented in the target system. The specification structuring is important for the translation process as well as the resulting design in order to associate the generated tests with requirements that must be mapped to the decisions in an implementation.

1.2 Organization of paper

Section 2 provides an overview of the SCR model and an example specification. Section 3 describes the T-VEC model and its relationship to the SCR model and language constructs. Section 4 provides an overview of the T-VEC test vector generation mechanisms, a description of the test vectors, and their relationship to the example specification. Section 5 summarizes the analysis results and the requirements for an SCR-to-T-VEC translator.

2. SCR model and example specification

This section presents an overview of the SCR specification constructs to support the T-VEC mapping discussion that is provided in Section 3. A more complete formal description of the SCR model can be found in [HJL96].

The Four Variable Model [PM91; Sch90] shown in Figure 1 provides a conceptual basis for describing the artifacts that are represented in an SCR specification.

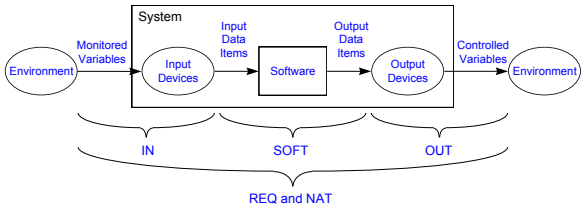

Figure 1. Four variable model

Monitored and *controlled* variables represent environmental quantities. REQ and NAT relations specify the required system behavior in terms of monitored and controlled variables. **NAT** defines the set of possible values; it captures any constraints on behavior imposed by physical laws. **REQ** defines the additional constraints imposed by the system to be built. Monitored variables must be mapped through input devices where they are represented as *input data items*; the **IN** relation specifies this mapping. Controlled variables are mapped as characterized by the **OUT** relation from *output data items*. The SCR model can describe system requirements or software requirements. As described in [HJL96], the term ***input variable*** is used to represent a monitored variable or input data item, and an ***output variable*** is used to a represent a controlled variable or output data item.

There are four other constructs that are used in the specification; these are modes, terms, conditions, and events. A *mode class* is a state machine, where related system states are called system modes and the transitions of the state machine are characterized by events. A *term* is any function in input variables, modes, or other terms. A *condition* is a predicate characterizing a system state. An *event* occurs when any system entity changes value.

2.1 Safety injection system example

The following is an example system that is presented in [HLK95; HJL96]. The specification is for

a Safety Injection control system. The system uses three sensors to monitor water pressure and adds coolant to the reactor core when the pressure falls below some threshold. The system operator blocks Safety Injection by turning on a Block switch and resets the system after blockage by turning on a Reset switch. Water pressure and the Block and Reset switches are represented as input variables; Safety Injection is an output variable. The specifications of type and constant definitions are shown in Tables 1 and 2. The input and output variable specifications are shown in Tables 3 and 4.

Table 1. Type dictionary

Name	Base Type	Units	Legal Values	Comment
Pressure	Integer	psi	[0,2000]	
Switch	Enumerated	N/A	ON, OFF	

Table 2. Constant dictionary

Name	Type	Value	Comment
Low	Pressure	900	
Permit	Pressure	1000	

Table 3. Input variable dictionary

Name	Type	Initial Value	Accuracy	Physical Interpretation
Block	Switch	OFF	N/A	
Reset	Switch	ON	N/A	
WaterPres	Pressure	14	0.05%	

Table 4. Output variable declaration

Name	Type	Initial Value	Accuracy	Physical Interpretation
Safety_Injection	Switch	OFF	N/A	

Table 5 defines a mode class representing pressure states. The mode class M_Pressure is an abstract model of the input variable WaterPres. At any given time, the system must be in one of three modes: TooLow, Permitted, or High.

Table 5. Mode class dictionary

Name	Modes	Initial Mode	Comment
M_Pressure	TooLow, Permitted, High	TooLow	

Table 6 defines a term Overridden. The term Overridden is true if Safety Injection is blocked, false otherwise.

Table 6. Term dictionary

Name	Type	Initial Value	Accuracy	Comment
Overridden	Boolean	TRUE	N/A	

The M_Pressure mode transition function, shown in Table 7, specifies the events that transition the system into the three different modes. For example, at the time when WaterPres becomes greater than or equal to Low (i.e., 900) then M_Pressure transitions from the state TooLow to Permitted.

Table 7. Mode transition function for M_Pressure

Source Mode	Events	Destination Mode
TooLow	@T(WaterPres >= Low)	Permitted
Permitted	@T(WaterPres < Low)	TooLow
Permitted	@T(WaterPres >= Permit)	High
High	@T(WaterPres < Permit)	Permitted

Tables 8 and 9 specify the overall behavior of the system in terms of modes, events, and terms. An example interpretation of Table 9 is: when the Mode is TooLow and Overridden is TRUE, then the value of Safety Injection is OFF. Table 8 specifies the function for the term Overridden; Overridden becomes TRUE when the mode is either TooLow or Permitted at the time point when Block becomes ON when Reset is OFF.

Table 8. Ideal value function for Overridden

Name	Mode Class	
Overridden	M_Pressure	
Mode	**Events**	
High	Never	@T(Inmode)
TooLow, Permitted	@T(Block=ON) WHEN Reset = OFF	@T(Inmode) OR @T(Reset = ON)
Overridden=	TRUE	FALSE

Table 9. Ideal value function for Safety Injection

Name	Mode Class	
Safety_Injection	M_Pressure	
Mode	**Events**	
High, Permitted	TRUE	FALSE
TooLow	Overridden	Not Overridden
Safety_Injection=	OFF	ON

3. T-VEC model and SCR mapping

This section focuses on T-VEC's specification models and how SCR specifications can be mapped into the T-VEC specification language. This focus helps describe the fundamental specification concepts that are relevant to test vector generation.

3.1 T-VEC models

T-VEC specifications are based on two models: a structural model and a requirement specification model. The ***structural model*** is a means for managing complexity using hierarchical relationships and for packaging common specification elements for reuse. The ***requirement specification model*** provides the basis for identifying and organizing the functional requirements for a subsystem. The formal definition of a functional requirement is the basis for the requirement specification model [Bus88; BB86]:

> ***the set of all functional relationships, for all points of temporal relevance, for a given output object***

Given a set of boundaries for a software system, the requirements are defined in terms of the syntactic structure and semantic values associated with the given input/output space. Figure 2 relates the functional requirements model to the precondition and postcondition model of Hoare [Hoa69]. A T-VEC subsystem is defined by: *outputs*, *inputs*, *functional relationships*, and *relevance predicates*. A ***functional relationship*** characterizes an object of the output space as a function of the inputs with respect to a relevance predicate. A ***relevance predicate*** groups all the precondition constraints associated with each functional relationship. Relevance predicates characterize data and temporal constraints on the objects of the input space.

A software system specification is captured in a set of related subsystems called a ***project***. The elements of a subsystem are defined as follows:

- SS is a subsystem that contains the following sets: {I, O, FR, RP, LS}
- I is the set of elements I_j of the input space, for all i,j $I_{i,j} \subset I_i \subset I$, where any $I_{i,j}$ is a subcomponent of I_i, and I is the Cartesian product of the set of all inputs.
- Similarly O is the set of elements O_j of the output space, where for all i,j $O_{i,j} \subset O_i \subset O$.
- FR is a set of functional relationships where FR_i defines the mapping $O_j = FR_i (I_k)$ with respect to a constraint called the relevance predicate $RP_i(I_m)$, where $I_m \subset I_k$ and $O_j \subset O$.

Functional relationship expressions are specified in terms of ***primitive operators***: bit operations, assignment, addition, subtraction, multiplication, division, exponentiation, absolute value, log, and trigonometric functions. Subsystems can be treated like functions, even if the subsystem specifies the requirements for complex objects; therefore, subsystems can be referenced within a functional relationship or in a relevance predicate. A functional relationship can also be expressed in terms of a *forall* operator when specifying a relationship governing some or all elements of a specified range of array elements.

- LS is a set of parameterized Boolean-valued statements that define constraints used in a relevance predicate. A ***logic structure*** defines constraints on the parameters in terms of connected conditions or logic structures using ∧, ∨, and ¬. A ***condition*** is a statement r φ s where φ ∈{=, ≠, >, <, ≥, ≤}; r ∈ I; and s is a variable, constant, arithmetic expression or functional relationship of another subsystem in the project.
- RP is a set of disjunctions of Boolean-valued statements. RP_i is of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_m$, p_i is a Boolean-valued statement r φ v where φ ∈{=, ≠, >, <, ≥, ≤}; r ∈ I; if p_i is a logic structure it can be referenced in the positive sense as p_i or negated as NOT:p_i.

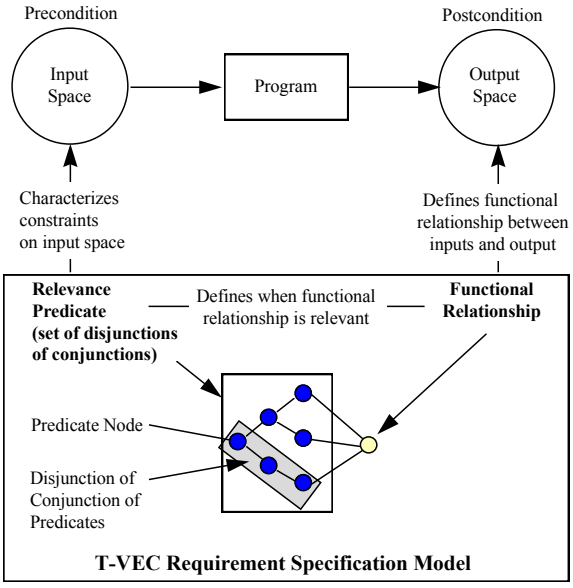

Figure 2. Relationship between T-VEC requirement specification model and precondition/postcondition model

The structural model defines the relationship between subsystems of a project. Subsystems can be related hierarchically such that $SS_{i,j}$ inherits I, O, and LS from SS_i. Any SS_i can reference input (I) and output (O) definitions. A subsystem can also reference functional relationships or logic structures from subsystems in the project by passing the input and output state space as a parameter.

Figure 3 shows an annotated T-VEC linear form[2] specification for the first version of the Safety Injection system. The labels on the left side of the figure are used later in the paper to explain the specification constructs. The system is specified with a project in one subsystem that contains the mode transition function for M_Pressure and the ideal functions for Overridden and Safety Injection. Version 2 of the example, shown in Figures 6 and 7, is specified in one project containing two subsystems where Overridden is

[2] The T-VEC linear form language specification can be found on the web at address http://www.crosslink.net/~blackbur/tvec_lrm.htm.

specified in a "lower-level" subsystem that is referenced by the Safety Injection subsystem.

3.2 Specification mapping goal

The SCR model is based on a finite state automaton, and T-VEC is a logic specification. The SCR model defines a system state in terms of entities, a condition as a predicate on the system state, and an input event as a change in an input variable [HJL96]. Modes represent event transitions over time. As shown in Figure 2, a T-VEC specification is defined in terms of the old state, characterized by the precondition, and the new state that is characterized by the postcondition. The relevance predicate specifies the constraints on input variables in the old state, and the functional relationship specifies the expected output for the new state. This directly supports the concept of an SCR condition.

A functional relationship that is related to an SCR event can depend on both the old state and new state of the specification entities, as described in [HJL96]. This means that a translation must create two instances of each input variable, term or mode, one for the old state and another for the new state. The relevance predicate characterizes the constraints associated with the old and new state; this means the relevance predicate is equivalent to the next-state relation for a state transition system. In the T-VEC specification, variables have a 1 or 2 suffix added to their name to indicate the association with the old state or new state. Variable definitions examples are shown in Figure 3.

The goal of the translation process is to map every SCR functional expression of an output variable to a T-VEC functional relationship of an output variable with respect to a set of predicates on the input variables. These predicates correspond to conditions, events, and corresponding mode transitions. Events must be reduced to predicates of input variables or terms for the old state and new state (i.e., the one before the event and the one after the event).

3.3 Specification construct mappings

Tables 10, 11, and 12 summarize the mapping between the elements of the SCR and T-VEC models in terms of their specification constructs. Table 10 shows that T-VEC inputs map to SCR input variables, and the outputs map to SCR output variables. SCR terms can be input or output variables, and mode classes are inputs. Both T-VEC and SCR support constant and type definitions. Both methods allow data to be specified in terms of base and derived types. Table 11 shows the type definition mappings; in T-VEC, numeric subranges or enumeration constants are described in constraints, and the analogous concepts are characterized as legal values in the SCR method. Both methods support accuracy and representation. Initial values can be specified in the SCR. As a logic specification, T-VEC does not explicitly represent state; any specific data value must be specified in terms of a predicate on the input variables as it relates to a functional relationship.

Table 10. Variable definition mapping

	Object Definitions			
T-VEC	Inputs Variable	Output Variable	Constant	Type
SCR	Input Variable, Terms, Mode Class	Output Variable, Terms	Constant	Type

Table 11. Type definition mapping

	Type Definitions				
T-VEC	Base or Derived Types	Constraint	Accuracy	Data Representation	<none>
SCR	Base or Derived Types	Legal Values	Accuracy	Physical Interpretation	Initial Value

Table 12 summarizes behavioral specification mappings. T-VEC functional requirements map to SCR ideal functions. T-VEC functional relationships map to SCR functional expressions. Relevance predicates map to modes, events, and conditions. SCR modes are defined in mode transition functions. SCR condition tables are not shown in the example specification.[3]

Table 12. Behavioral specification mapping

	Behavioral Specifications				
T-VEC	Functional Req's.	Functional Relationships	Relevance Predicate		
SCR	Ideal Function	Functional Expression	Modes	Events	Conditions

[3] An SCR condition maps directly into a T-VEC relevance predicate because it characterizes constraints on the input space at a time point prior to the execution of a function. In T-VEC, conditions are represented as logic structures or simple predicates.

```
        SUBSYSTEM  safety_injection1 (WaterPres1, WaterPres2, Reset1, Reset2, Block1, Block2, Overridden1, Overridden2, Pressure1, Pressure2)
        {
          TYPE Switch              ISA ENUMERATION RANGE {OFF=0, ON=1};
          TYPE Pressure            ISA INTEGER RANGE {0..2000};
          TYPE M_Pressure_Mode     ISA ENUMERATION RANGE {TooLow=0, Permitted=1, High=2};
          CONSTANT Low             ISA Pressure VALUE 900;
          CONSTANT Permit          ISA Pressure VALUE 1000;
          VARIABLE Safety_Injection ISA Switch;
          VARIABLE WaterPres1      ISA Pressure;
          VARIABLE WaterPres2      ISA Pressure;
          VARIABLE Reset1          ISA Switch;
          VARIABLE Reset2          ISA Switch;
          VARIABLE Block1          ISA Switch;
          VARIABLE Block2          ISA Switch;
          VARIABLE Overridden1     ISA BOOLEAN;
          VARIABLE Overridden2     ISA BOOLEAN;
          VARIABLE Pressure1       ISA M_Pressure_Mode;
          VARIABLE Pressure2       ISA M_Pressure_Mode;

LS1       LOGIC STRUCTURE At_T_Reset_TooLow (Pressure1:M_Pressure_Mode, Reset1:Switch, Reset2:Switch)
            CONSTRAINT(Pressure1 = TooLow and Reset1 = OFF and Reset2 = ON);

LS2       LOGIC STRUCTURE At_T_Reset_Permitted (Pressure1:M_Pressure_Mode, Reset1:Switch, Reset2:Switch )
            CONSTRAINT(Pressure1 = Permitted and Reset1 = OFF and Reset2 = ON);

LS3       LOGIC STRUCTURE At_T_Inmode_High (Pressure1:M_Pressure_Mode, Pressure2:M_Pressure_Mode)
            CONSTRAINT (Pressure2 = High and Pressure1 != High);

LS4       LOGIC STRUCTURE At_T_Inmode_TooLow_Permitted (Pressure1:M_Pressure_Mode, Pressure2:M_Pressure_Mode)
           CONSTRAINT ((Pressure2 = Permitted or Pressure2 = TooLow) and
                       (Pressure1 != Permitted and Pressure1 != TooLow));

LS5       LOGIC STRUCTURE At_T_Block_On (Reset1:Switch, Reset2:Switch, Block1:Switch, Block2:Switch)
            CONSTRAINT (Block2 = ON and Block1 = OFF and Reset1 = OFF and Reset2 = OFF);

LS6       LOGIC STRUCTURE Overridden_Term (Overridden2, Pressure1, Pressure2, Reset1, Reset2, Block1, Block2)
            CONSTRAINT
LS6.1           Overridden2 = false and
LS6.1.1                                  (At_T_Reset_TooLow(Pressure1, Reset1, Reset2)
LS6.1.2                                OR At_T_Reset_Permitted(Pressure1, Reset1, Reset2)
LS6.1.3                                OR At_T_Inmode_High(Pressure1, Pressure2)
LS6.1.4                                OR At_T_Inmode_TooLow_Permitted(Pressure1, Pressure2))
LS6.2        OR Overridden2 = true and
LS6.2.1                                  (Pressure1 = TooLow and At_T_Block_On(Reset1, Reset2, Block1, Block2)
LS6.2.2                                OR (Pressure1 = Permitted and At_T_Block_On(Reset1, Reset2, Block1, Block2)));

LS7       LOGIC STRUCTURE M_Pressure (WaterPres1, WaterPres2, Pressure2, Pressure1)
            CONSTRAINT
LS7.1         Pressure2 = TooLow and Pressure1 = Permitted
                          and WaterPres2 < Low and WaterPres1 >= Low and WaterPres1 < Permit
LS7.2       OR Pressure2 = High and Pressure1 = Permitted
                         and WaterPres2 >= Permit and WaterPres1 < Permit and WaterPres1 >= Low
LS7.3       OR Pressure2 = Permitted and    (Pressure1 = TooLow and WaterPres2 >= Low  and WaterPres2 < Permit and WaterPres1 < Low
                                          OR Pressure1 = High and WaterPres2 < Permit and WaterPres2 >= Low and WaterPres1 >= Permit)
LS7.4       OR Pressure2 = Pressure1 and    ((Pressure1 = TooLow and WaterPres2 < Low and WaterPres1 < Low)
                                          OR  (Pressure1 = Permitted and WaterPres2 < Permit and WaterPres1 < Permit
                                                                     and WaterPres2 >= Low and WaterPres1 >= Low)
                                          OR  (Pressure1 = High and WaterPres2 >= Permit and WaterPres1 >= Permit));
          FUNCTIONAL REQUIREMENTS
          LEVEL {
FR.0          RELATIONSHIP produce Safety_Injection;
              RELEVANCE PREDICATE {
RP.0            DISJUNCTION {M_Pressure};
              }
            LEVEL {
FR.1          RELATIONSHIP Safety_Injection = ON;
              RELEVANCE PREDICATE {
RP.1.1          DISJUNCTION {Pressure2 = TooLow, Overridden2 = false, Overridden_Term};
RP.1.2          DISJUNCTION {Pressure2 = TooLow, Overridden2 = false, Overridden1 = false, NOT:At_T_Reset_TooLow, NOT:At_T_Reset_Permitted,
                             NOT:At_T_Inmode_High, NOT:At_T_Inmode_TooLow_Permitted, NOT:At_T_Block_On};
              }
FR.2          RELATIONSHIP Safety_Injection = OFF;
              RELEVANCE PREDICATE {
RP.2.1          DISJUNCTION {Pressure2 = High};
RP.2.2          DISJUNCTION {Pressure2 = Permitted};
RP.2.3          DISJUNCTION {Pressure2 = TooLow, Overridden2 = true, Overridden_Term};
RP.2.4          DISJUNCTION {Pressure2 = TooLow, Overridden2 = true, Overridden1 = true, NOT:At_T_Reset_TooLow, NOT:At_T_Reset_Permitted,
                             NOT:At_T_Inmode_High, NOT:At_T_Inmode_TooLow_Permitted, NOT:At_T_Block_On};
          }  }  }  }
```

Figure 3. Safety injection example version 1 in T-VEC linear form

3.4 Mapping mode transition functions

An SCR mode transition function defines a mapping from one mode to another based on the occurrence of an event. In T-VEC, the mode transition function is mapped into a logic structure. See LS7 in Figure 3. The source mode, `Pressure1` is defined as part of a relevance predicate old state, and the event specifies the condition for the transition to the destination mode `Pressure2`. This translation is based on the specification given in standard logic in [HJL96], with some minor exceptions. In [HJL96], there is an assumption that limits the rate of change of the input variable WaterPres. This assumption precludes the transitions from TooLow to High and from High to TooLow. The constraints associated with this assumption have been formalized in LS7. The transition from LS7.1 specifies the transition from Permitted to TooLow; LS7.2 specifies the transition from Permitted to High; LS7.3 specifies both transitions as disjunctions, one from TooLow to Permitted and the other from High to Permitted. LS7.4 specifies the case where there is no transition.

3.5 Mapping ideal functions of terms

Overridden is an ideal function for a term variable. The term Overridden is used in the Safety Injection ideal function as shown in Figure 4. In SCR, an ideal function for a term variable modularizes a specification. The term variable is not an output of the system, but it is used in the constraint of the Safety Injection ideal function. Overridden can be mapped to the T-VEC model in two ways: as a set of predicates represented in a logic structure or as a lower-level subsystem that has a Boolean output associated with the value of the function. For version 1, the function Overridden is translated as a logic structure that is referenced within the Safety Injection relevance predicate. Figure 3 shows the representation of the Overridden ideal function as a logic structure LS6. The two disjunctions, LS6.1 and LS6.2, map to the two outputs of the ideal function Overridden. The nested disjunctions, LS6.1.1 through LS6.1.4, are required when Overridden2 is false, and disjunctions LS6.2.1 and LS6.2.2 are required when Overridden2 is true. These specifications also illustrate the use of parameterized logic structures, which are reused in the negated sense to characterize the states when Overridden2 does not change state. This situation is discussed in the next section.

One case is not reflected in the translation; Table 8 specifies that when the mode is High there is no event that should set Overridden to true. From a test vector generation perspective, there is no way to produce a test vector with externally visible results for this case. The verification of this requirement requires manual analysis of the target implementation.

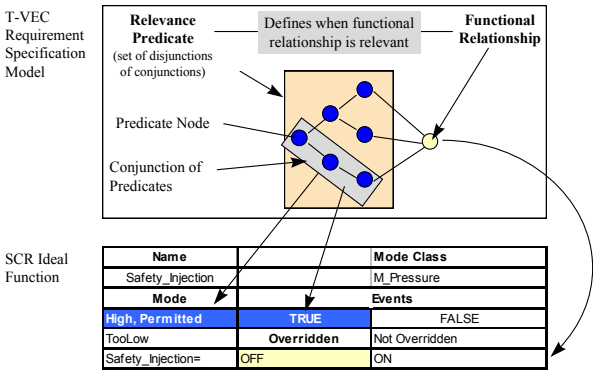

Figure 4. Mapping between T-VEC requirement specification and SCR ideal function

3.6 Functional relationships

Figure 4 shows the correspondence between a T-VEC functional relationship and an SCR functional expression. The functional requirement output is Safety Injection, and it is based on the mode class M_Pressure. The functional relationship FR.0 is associated with one disjunction for the relevance predicate RP.0, which states that M_Pressure is always relevant to any functional relationship for Safety Injection. Figure 3 shows how the specification characterizes the functional relationship for each functional expression of the SCR specification (one when Safety Injection is ON and the other when it is OFF); they are labeled FR.1 and FR.2. The functional relationship FR.1 has one disjunction RP1.1. The conditions of this disjunction are consistent with the ideal function for Safety Injection shown in Table 9; it specifies that `Pressure2 = TooLow` and `Overridden2 = false` in the context of the logic structure Overridden_Term (i.e., the logic structure representation of Overridden). In addition, any disjunction must also satisfy the disjunctions of RP.0, which characterizes the constraints of M_Pressure. Similarly the relevance predicate for FR.2 has three disjunctions, RP.2.1, RP.2.2, RP.2.3, that correspond to the cases when Safety Injection is OFF.

The relevance predicates RP.1.2 and RP.2.1 have been separated to clearly identify the treatment of the case where Overridden does not change state. These are the cases when both `Overridden2 = Overridden1` and all of the constraints associated

with the events specified in Overridden_Term do not occur. This is reflected by the conjunction of the negated logic structure:

```
NOT:At_T_Reset_TooLow and
NOT:At_T_Reset_Permitted and
NOT:At_T_Inmode_High and
NOT:At_T_Inmode_TooLow_Permitted and
NOT:At_T_Block_On
```

4. Resulting test vectors

This section describes the test vectors that were generated for two different versions of the specification:

1. Overridden specified as a constraint within the Safety Injection subsystem (version 1)
2. Overridden specified as a lower-level subsystem (version 2 – described later in this section)

First, the linear form transformation and test vector generation concepts are described.

4.1 Specification compilation

A specification compiler transforms the linear form shown in Figure 3 into a logic specification represented in a Prolog-like language. During the process, syntax and semantic checks are performed to ensure that the resulting specification complies with the T-VEC models. The disjunctions in logic structures are expanded and a DeMorganization process is applied to logic structures preceded by the `NOT:` operator. When the compiled specification is loaded into the test vector generator, each functional relationship is associated with a set of disjunctions of conjunctions characterized by the "flattened" and DeMorganized logic structures within a subsystem. Each disjunction is referred to as a domain convergence path (DCP).

4.2 Test vector generation concepts and mechanisms

T-VEC is an ***oracle/error-based*** testing mechanism based on Richardson's et. al [ROT89] classification of specification-based testing approaches; such approaches extend implementation-based testing techniques to formal specifications. The T-VEC test selection mechanisms are related to implementation-based testing concepts and strategies.

Using Zeil's [Zei89] modified version of Howden's [How76] definitions: a ***computation error*** occurs when the correct path through the program is taken, but the output is incorrect due to faults in the computation along the path. A ***domain error*** occurs when an incorrect output is generated due to executing the wrong path through a program. Based on the assumption that there is a strong correlation between predicates in the specification and path control conditions in the program, the test selection strategies are discussed in terms of domain testing theory concepts. White and Cohen [WC80] proposed ***domain testing theory*** as a strategy for selecting test points to reveal domain errors. It is based on the premise that if there is no coincidental correctness, then test cases that localize the boundaries of domains with arbitrarily high precision are sufficient to test all the points in the domain.

T-VEC selects test data for subdomains of an input space based on the constraints of a DCP. The DCP predicates should map to the path conditions in a corresponding program.

A subdomain convergence algorithm is used to determine a DCP subdomain. If a nonempty subdomain exists for a DCP, then the input values associated with a test point are selected for the borders of the subdomain. A ***border*** is defined by evaluating the predicates of a DCP for a set of input values. For example, test points for numeric objects are selected for both upper and lower domain boundary values. This results in test points for subdomain borders based on all low-bound values and high-bound input values that satisfy the DCP predicate evaluation. Some inputs to the functional relationship are not constrained by the DCP predicates. For each test point derived from DCP predicates, there are additional test points derived for unconstrained inputs not referenced in the DCP based on all domain boundary value combinations (e.g., low bound and high bound for numeric objects, sets for enumerated variable). By selecting the extreme value combinations, it is possible to detect computation errors in the output calculation. This test selection strategy is used to detect computation errors or show that unconstrained inputs do not affect the output for a program path.

The functional relationship is applied to each input value set to determine the expected output value. The value is checked against the subrange specification of the output variable; if the value is within the specified range, a test vector is produced that includes the inputs, input types and representation information, the expected output with its type information, and the DCP.

4.3 Resulting test vectors

The version 1 specification resulted in 178 test vectors that were generated in single vector mode (see Section 4.4). There were 18 test vectors generated for RP.1.1, RP.2.1, RP.2.2, and RP.2.3 that correspond to

the cases when Overridden changes; these vectors are shown in Table 13. The remaining 160 test vectors were associated with relevance predicates RP.1.2 and RP.2.4, which specify the states where there is no change in Overridden. These specifications are important for testing to ensure that the system stays in the specified mode when the inputs do not change. These tests are not shown due to space limitations.

Each row corresponds to one test vector. There are columns that identify the test vector number, the functional relationship (FR), the primary constraint of the relevance predicate (i.e., the prefix of the DCP), the convergence mode (e.g., low bound or high bound for numeric variables), and the values for the expected output (i.e., Safety Injection) and each input.

4.4 Vector generation modes

There are selectable modes for generating test vectors: multi, restricted, and single vector modes. The test point selection also depends on the relevance predicate constraints for each functional relationship. Table 14 shows the relationships between the resulting test points in each mode, with respect to a constraint. Given two variables x and y, each with a subrange of –10 to 10, and a functional relationship `z = x + y;` assume there are two relevance predicate constraints: 1) `x > y`, and 2) TRUE (i.e., there are no constraints on the variables x and y other than their subranges). For constraint 1, test points are selected based on the subdomain constraints for high bound and low bound combinations. Based on the assumption stated in Section 4.2 (i.e., the DCP predicates should map to the path conditions in a corresponding program), this heuristic has been effective in producing a minimal number of test cases to exercise each decision in a program with both high bound and low bound cases. For unconstrained variables, the following rules apply (i.e., the row labeled TRUE in Table 14):

- Multi: All combinations of the inputs based on the variable's range are selected as test points (this is effective in detecting computation errors).
- Restricted: Each high bound and low bound of each variable as well as the high bound combination (as the number of inputs increases this number will result in significantly less tests than multi-vector mode).
- Single vector mode: Only the high bound test points will be selected (this is effective in detecting computation errors, like overflows).

Table 14. Vector generation mode example

	Multi		Restricted		Single	
Constraint	x	y	x	y	x	y
1) x > y	-9	-10	-9	-10	-9	-10
	10	9	10	9	10	9
2) TRUE	10	10	10	10	10	10
	10	-10	10	-10		
	-10	10	-10	10		
	-10	-10				

Table 13. Test vector summary for version 1 – Safety Injection

Vector #	FR	RP	Convergence Mode	Safety Injection	Overridden2	Pressure1	Pressure2	WaterPres1	WaterPres2	Reset1	Reset2	Block1	Block2
1	1	RP__1<<1>>	Low Bound	ON	FALSE	TooLow	TooLow	0	0	OFF	ON	-	-
2	1	RP__1<<2>>	Low Bound	ON	FALSE	Permitted	TooLow	900	0	OFF	ON	-	-
3	1	RP__1<<1>>	Hi Bound	ON	FALSE	TooLow	TooLow	899	899	OFF	ON	-	-
4	1	RP__1<<2>>	Hi Bound	ON	FALSE	Permitted	TooLow	999	899	OFF	ON	-	-
5	2	RP__2<<1>>	Low Bound	OFF	FALSE	Permitted	High	900	1000	OFF	OFF	OFF	OFF
6	2	RP__2<<1>>	Low Bound	OFF	FALSE	High	High	1000	1000	OFF	OFF	OFF	OFF
7	2	RP__2<<2>>	Low Bound	OFF	FALSE	TooLow	Permitted	0	900	OFF	OFF	OFF	OFF
8	2	RP__2<<2>>	Low Bound	OFF	FALSE	High	Permitted	1000	900	OFF	OFF	OFF	OFF
9	2	RP__2<<2>>	Low Bound	OFF	FALSE	Permitted	Permitted	900	900	OFF	OFF	OFF	OFF
10	2	RP__2<<3>>	Low Bound	OFF	TRUE	TooLow	TooLow	0	0	OFF	OFF	OFF	ON
11	2	RP__2<<4>>	Low Bound	OFF	TRUE	Permitted	TooLow	900	0	OFF	OFF	OFF	ON
12	2	RP__2<<1>>	Hi Bound	OFF	FALSE	Permitted	High	999	2000	OFF	OFF	OFF	OFF
13	2	RP__2<<1>>	Hi Bound	OFF	FALSE	High	High	2000	2000	OFF	OFF	OFF	OFF
14	2	RP__2<<2>>	Hi Bound	OFF	FALSE	TooLow	Permitted	899	999	OFF	OFF	OFF	OFF
15	2	RP__2<<2>>	Hi Bound	OFF	FALSE	High	Permitted	2000	999	OFF	OFF	OFF	OFF
16	2	RP__2<<2>>	Hi Bound	OFF	FALSE	Permitted	Permitted	999	999	OFF	OFF	OFF	OFF
17	2	RP__2<<3>>	Hi Bound	OFF	TRUE	TooLow	TooLow	899	899	OFF	OFF	OFF	ON
18	2	RP__2<<4>>	Hi Bound	OFF	TRUE	Permitted	TooLow	999	899	OFF	OFF	OFF	ON

The internal form of a test vector is shown in Figure 5. The first line indicates the functional relationship and relevance predicate. The <<1>> indicates that this is the first Safety Injection functional relationship; each predicate disjunction is identified by a unique number using this notation. The OUTPUT section includes one object, Safety Injection. Each output also includes the associated type, data representation (e.g., 32 bits) physical value (1), and enumeration constant (ON). This is followed by the INPUTS section that lists all the input objects. Each input object has the same information as the output object. Finally, the relevance predicate disjunction (i.e., DCP) that was used in the vector generation process is delineated by the START_JUSTIFICATION and END_JUSTIFICATION keywords. The justification path is part of the input to the coverage analysis process.

4.5 Specification-based coverage analysis

After test vectors are generated, a check is performed to ensure that each specification corresponding to a DCP has at least one test vector. The T-VEC specification-based coverage analysis tool identifies specification inconsistencies that occur when there is not a complete mapping between the generated test vectors and the set of all DCP combinations in the compiled specification for a subsystem. Specification inconsistencies can result when:

- The convergence process cannot determine an input subdomain for a DCP because there is an inconsistent set of predicates in the DCP.
- The expected output value, computed using the functional relationship with the input test values, is not correct with respect to its subrange specification.

The coverage analyzer was used to check the resulting test vectors against the compiled specification. The coverage analysis results helped identify some translation guidelines that were not applied to the version 1 translation. By combining ideal functions into one subsystem, there are system states for the ideal function Safety Injection that cannot be satisfied by all combinations of cases when Overridden = false. For the Safety Injection requirement when Pressure2 = TooLow and Overridden = false, there were two DCPs that were flagged by the coverage analyzer, when Pressure2 is required to be High or Permitted in the ideal function Overridden. The first case involves statements RP.1.1 (Safety Injection), RP.0 (M_Pressure), LS6.1.3 (Overridden) shown in Figure 3. The expanded conditions are:

```
Pressure2 = TooLow, Overridden2 = False,
Pressure2 = High and Pressure1 != High
```

In this case, Pressure2 cannot be TooLow and High. The TooLow condition comes from the Safety Injection ideal function, and the Pressure2 = High is specified in the Overridden ideal function @Inmode(High). The second set of conditions is related to the statements RP.1.1, RP.0, and LS6.1.4. The expanded conditions are:

```
Pressure2 = TooLow and Overridden2 = False and
Pressure1 != Permitted and Pressure1 !=  TooLow
and Pressure2 = Permitted
```

In this case, Pressure2 cannot be TooLow and Permitted. Similarly, the TooLow condition comes from the Safety Injection ideal function, and the Pressure2 = Permitted is specified in Overridden.

4.6 Version 2: Hierarchy of specifications

Guidelines to localize the constraints of an ideal function to a subsystem were applied to the second translation; this affects the way tests are generated and coverage analysis is performed. The specification was translated into a hierarchy of two subsystems: one for Safety Injection and one for Overridden. The structure of the T-VEC subsystems maps to the relationship of the ideal functions of an SCR specification.

The translated specifications are shown in Figures 6 and 7. Figure 6 is almost identical to Figure 3; the type, variable, constant sections have been removed, as well as the M_Pressure logic structure, as annotated in the figure. The key difference is the logic structure labeled LS_2.1. The Overridden_Term logic structure references a subsystem named Overridden, instead of expanding the ideal function in the logic structure. Overridden inherits data, type, and logic structure information from Safety Injection, as annotated in Figure 6.

```
safety_injection1<<1>>,  RP__1<<1>>
OUTPUT
Safety_Injection  ENUMERATION  32  1 ON
INPUTS
WaterPres1  INTEGER  32  0
WaterPres2  INTEGER  32  0
Reset1  ENUMERATION  32  0 OFF
Reset2  ENUMERATION  32  1 ON
Overridden2  BOOLEAN  32  0 FALSE
Pressure1  ENUMERATION  32  0 TooLow
Pressure2  ENUMERATION  32  0 TooLow
START_JUSTIFICATION
solution number -> 1
safety_injection1, safety_injection1_FR__1,
enter_cv_mode<<2>>, cv_safety_injection1_RP__1,
safety_injection1_RP__1, safety_injection1_RP__0,
M_Pressure<<4>>, M_Pressure.1, At_T_Reset_TooLow,
exit_cv_mode
END_JUSTIFICATION
```

Figure 5. T-VEC internal form of test vector

In Figure 7, functional relationship FR_2.0 is associated with the Overridden2 variable for the ideal function Overridden. The relevance predicate RP_2.0 is related to the M_Pressure mode class as it was in version 1. FR_2.1 maps to the TRUE output of the function, with respect to constraints RP_2.1.1 and RP_2.1.2. Similarly, FR_2.2 maps to the FALSE output of Overridden with respect to constraints RP_2.2.1 through RP_2.2.4. These conditions map directly to the events in Table 8. Finally, RP_2.1.3 and RP_2.2.5 are associated with the situation when Overridden does not change state, using the negation of state change events defined in the logic structures.

4.7 Version 2: Hierarchical test vector generation and coverage analysis

A unique mechanism of T-VEC supports the generation of test vectors for a hierarchy of specifications, without regenerating all test vectors for each referenced lower-level subsystem. When any function, like Safety Injection, references another function, like Overridden, the test vector generator treats Overridden like a primitive operator. This mechanism precludes the combinatorial explosion associated with tests generated from the combination of constraints in a hierarchy of subsystems.

The T-VEC coverage analyzer tool distinguishes between the specifications in the parent subsystem and any child subsystem that is used to support the functionality of a parent. In the case of Safety Injection, the coverage analyzer checked that a function reference was made to Overridden, rather than checking all the constraints associated for Overridden.

4.8 Version 2: Resulting test vectors and coverage analysis

Test vectors were regenerated for both subsystems. Table 15 shows the test vectors for Safety Injection. Table 16 shows 24 vectors associated with the FR_2.1 and FR_2.2 relevance predicates with state changes as specified by the events for the ideal function. There were 468 total test vectors for Overridden; however, 444 of these vectors were related to the conditions associated with the state not changing, as specified by RP_2.1.3 and RP_2.2.5 in Figure 7.

The coverage analysis check was performed for version 2, and the test vectors for Safety Injection and Overridden covered all DCP combinations of the compiled specification. However, a NAT constraint specified in M_Pressure did cause the coverage analyzer to flag a constraint in Overridden. M_Pressure limits the amount that WaterPres can change during one state transition. The constraint associated with RP_2.2.4, which references the logic structure labeled LS_2.1 specifies that `Pressure1 != Permitted ∧ Pressure1 != TooLow`; this implies that `Pressure1 = High`, and `Pressure2 = Permitted ∨ Pressure2 = TooLow`. Based on the constraints of M_Pressure, if `Pressure1 = High`, the only permitted transition is `Pressure2 = Permitted`. This is consistent with the NAT constraint.

```
        SUBSYSTEM  safety_injection2 (WaterPres1, WaterPres2, Reset1, Reset2, Block1, Block2, Overridden1, Overridden2, Pressure1, Pressure2)
        {
   *** Type, constant, and variable specifications identical to version 1 shown in Figure 3

          LOGIC STRUCTURE Overridden_Term (WaterPres1, WaterPres2, Reset1, Reset2,
                                           Block1, Block2, Pressure1, Pressure2, Overridden1, Overridden2)
LS_2.1      CONSTRAINT Overridden2 = Overridden(WaterPres1, WaterPres2, Reset1, Reset2,
                                                Block1, Block2, Pressure1, Pressure2, Overridden1, Overridden2);
  *** M_Pressure logic structure specification identical to version 1 shown in Figure 3
          FUNCTIONAL REQUIREMENTS
          LEVEL {
            RELATIONSHIP produce Safety_Injection;
              RELEVANCE PREDICATE {
                  DISJUNCTION {M_Pressure};
              }
            LEVEL {
              RELATIONSHIP Safety_Injection = ON;
                RELEVANCE PREDICATE {
                  DISJUNCTION {Pressure2 = TooLow, Overridden2 = false, Overridden_Term};
                }
              RELATIONSHIP Safety_Injection = OFF;
                RELEVANCE PREDICATE {
                  DISJUNCTION {Pressure2 = High};
                  DISJUNCTION {Pressure2 = Permitted};
                  DISJUNCTION {Pressure2 = TooLow, Overridden2 = true, Overridden_Term};
          } } } }
```

Figure 6. Safety Injection example version 2 in T-VEC linear form

```
        SUBSYSTEM  Overridden (WaterPres1, WaterPres2, Reset1, Reset2, Block1, Block2, Pressure1, Pressure2, Overridden1, Overridden2)
        {
          LOGIC STRUCTURE At_T_Reset_TooLow (Pressure1, Reset1, Reset2) CONSTRAINT (Pressure1 = TooLow and Reset1 = OFF and Reset2 = ON);
          LOGIC STRUCTURE At_T_Reset_Permitted (Pressure1, Reset1, Reset2) CONSTRAINT (Pressure1 = Permitted and Reset1 = OFF and Reset2 = ON);
          LOGIC STRUCTURE At_T_Inmode_High (Pressure1, Pressure2) CONSTRAINT (Pressure2 = High and Pressure1 != High);
LS_2.1    LOGIC STRUCTURE At_T_Inmode_TooLow_Permitted (Pressure1, Pressure2)
            CONSTRAINT ((Pressure2 = Permitted or Pressure2 = TooLow)
                         and
                        (Pressure1 != Permitted and Pressure1 != TooLow));
          LOGIC STRUCTURE At_T_Block_On (Reset1, Reset2, Block1, Block2)
            CONSTRAINT (Block2 = ON and Block1 = OFF and Reset1 = OFF and Reset2 = OFF);

          FUNCTIONAL REQUIREMENTS
          LEVEL {
FR_2.0      RELATIONSHIP produce Overridden2;
              RELEVANCE PREDICATE {
RP_2.0            DISJUNCTION {M_Pressure};
              }
            LEVEL {
FR_2.1        RELATIONSHIP Overridden2 = True;
                RELEVANCE PREDICATE {
RP_2.1.1          DISJUNCTION {Pressure1 = TooLow, At_T_Block_On};
RP_2.1.2          DISJUNCTION {Pressure1 = Permitted, At_T_Block_On};
RP_2.1.3          DISJUNCTION {Overridden1 = true, NOT:At_T_Reset_TooLow, NOT:At_T_Reset_Permitted,
                               NOT:At_T_Inmode_High, NOT:At_T_Inmode_TooLow_Permitted, NOT:At_T_Block_On};
                }
FR_2.2        RELATIONSHIP Overridden2 = False;
                RELEVANCE PREDICATE {
RP_2.2.1          DISJUNCTION {At_T_Reset_TooLow};
RP_2.2.2          DISJUNCTION {At_T_Reset_Permitted};
RP_2.2.3          DISJUNCTION {At_T_Inmode_High};
RP_2.2.4          DISJUNCTION {At_T_Inmode_TooLow_Permitted};
RP_2.2.5          DISJUNCTION {Overridden1 = false, NOT:At_T_Reset_TooLow, NOT:At_T_Reset_Permitted,
                               NOT:At_T_Inmode_High, NOT:At_T_Inmode_TooLow_Permitted, NOT:At_T_Block_On};
               } } } }
```

Figure 7. Overridden subsystem specification for version 2 in T-VEC linear form

5. Summary

This paper describes an effort to integrate the SCR formal methods approach with the T-VEC automatic test vector generator and specification analysis system. The results indicate the strong potential for mechanically translating SCR-style specifications into the T-VEC specifications to support automatic test vector generation. Two versions of an example SCR specification were manually translated into the T-VEC language. Test vectors were generated for the two versions using the T-VEC system. The relationships between the specifications and the resulting test vectors were analyzed. Two general guidelines for the translation process were identified:

1. An SCR mode transition function defines a mapping from one mode to another based on the occurrence of an event. Variables referenced in an SCR event operator must be expanded into two states when translated into the T-VEC model to adequately represent the states before and after the event. In T-VEC, the source mode is defined as a constraint on the input state space at some time point, and the event at some later time point leads to the destination mode. For test vector generation, each possible set of input variables must be tested to ensure that the software reacts correctly with respect to its specification. The translated specification must characterize the transition case for each mode event, as well as the case where there is no transition, to ensure that the system stays in the specified mode when the inputs do not change.
2. The structure of the T-VEC specification must map to the ideal functions of an SCR specification so that only those constraints directly associated with an ideal function are relevant to the test vector generation and coverage analysis process.

The need for mapping ideal functions to T-VEC subsystems was demonstrated. For version 1 of the specification, the Safety Injection and Overridden ideal functions were included in the same T-VEC subsystem; the analysis identified the need to localize the constraints of an ideal function to a subsystem; this affects the way tests are generated and coverage analysis is performed. In version 2, the ideal functions were represented as a hierarchy of subsystems, where

Overridden is a child subsystem to Safety Injection. Test vectors were generated and all valid constraints of the specifications were fully covered by test vectors.

Therefore, based on the results, every SCR ideal function must be translated into a T-VEC subsystem; this will result in a hierarchy of T-VEC subsystems that correspond to the logical hierarchy of SCR ideal functions. When T-VEC generates test vectors for a hierarchy of specifications it supports integration testing that covers the relationships between ideal functions. The tests for each subsystem should be injected into the target system using a bottom-up testing strategy. To adequately test the implementation, the structure of the implementation should correspond with the specification so that the DCP predicates map to the path conditions in a corresponding program. If all tests pass, there is a strong argument that all constraints in the specifications have a valid implementation in terms of the decisions guarding the computations that implement the functional expressions of an ideal function. This approach was demonstrated for real-world applications. T-VEC was used in two critical system developments within an industrial engineering organization for two systems that have been certified by the FAA [BB96]. To provide assurance that the implementation is complete and consistent with respect to the specification, this type of consistency is typically required for software developed to support FAA certifications [RTCA92]. The structural mapping from the specification to the implementation implies that the specifier should have some influence on the design of the resulting implementation if requirements-to-test traceability is a requirement of the verification process.

The combination of requirement specification methods and tools with automatic test generation technologies could significantly reduce the cost of verification and testing. T-VEC significantly reduced the verification cost by eliminating most of the manual testing effort on the last release of the MD90 Electrical Power System Variable Speed Constant Frequency system; there was a 6 to 1 reduction in time, effort and cost on the reverification of the system [BB96]. This experimental integration demonstrates the utility and benefits of such an approach and provides strong arguments for further advancing specification-based methods and tools.

Table 15. Test vector summary for version 2 – Safety Injection

Vector #	FR	RP	Convergence Mode	Safety Injection	Overridden2	Overridden1	Pressure1	Pressure2	WaterPres1	WaterPres2	Reset1	Reset2	Block1	Block2
1	1	RP__1<<1>>	Low Bound	ON	FALSE	FALSE	Permitted	TooLow	900	0	OFF	ON	OFF	OFF
2	1	RP__1<<1>>	Low Bound	ON	FALSE	FALSE	TooLow	TooLow	0	0	OFF	ON	OFF	OFF
3	1	RP__1<<1>>	Hi Bound	ON	FALSE	FALSE	Permitted	TooLow	999	899	OFF	ON	OFF	OFF
4	1	RP__1<<1>>	Hi Bound	ON	FALSE	FALSE	TooLow	TooLow	899	899	OFF	ON	OFF	OFF
5	2	RP__2<<1>>	Low Bound	OFF	FALSE	FALSE	Permitted	High	900	1000	OFF	OFF	OFF	OFF
6	2	RP__2<<1>>	Low Bound	OFF	FALSE	FALSE	High	High	1000	1000	OFF	OFF	OFF	OFF
7	2	RP__2<<2>>	Low Bound	OFF	FALSE	FALSE	TooLow	Permitted	0	900	OFF	OFF	OFF	OFF
8	2	RP__2<<2>>	Low Bound	OFF	FALSE	FALSE	High	Permitted	1000	900	OFF	OFF	OFF	OFF
9	2	RP__2<<2>>	Low Bound	OFF	FALSE	FALSE	Permitted	Permitted	900	900	OFF	OFF	OFF	OFF
10	2	RP__2<<3>>	Low Bound	OFF	TRUE	FALSE	Permitted	TooLow	900	0	OFF	OFF	OFF	ON
11	2	RP__2<<3>>	Low Bound	OFF	TRUE	FALSE	TooLow	TooLow	0	0	OFF	OFF	OFF	ON
12	2	RP__2<<1>>	Hi Bound	OFF	FALSE	FALSE	Permitted	High	999	2000	OFF	OFF	OFF	OFF
13	2	RP__2<<1>>	Hi Bound	OFF	FALSE	FALSE	High	High	2000	2000	OFF	OFF	OFF	OFF
14	2	RP__2<<2>>	Hi Bound	OFF	FALSE	FALSE	TooLow	Permitted	899	999	OFF	OFF	OFF	OFF
15	2	RP__2<<2>>	Hi Bound	OFF	FALSE	FALSE	High	Permitted	2000	999	OFF	OFF	OFF	OFF
16	2	RP__2<<2>>	Hi Bound	OFF	FALSE	FALSE	Permitted	Permitted	999	999	OFF	OFF	OFF	OFF
17	2	RP__2<<3>>	Hi Bound	OFF	TRUE	FALSE	Permitted	TooLow	999	899	OFF	OFF	OFF	ON
18	2	RP__2<<3>>	Hi Bound	OFF	TRUE	FALSE	TooLow	TooLow	899	899	OFF	OFF	OFF	ON

Table 16. Test vector summary for version 2 - Overridden

Vector #	FR	RP	Convergence Mode	Overridden2	Pressure1	Pressure2	WaterPres1	WaterPres2	Reset1	Reset2	Block1	Block2
1	1	RP__1<<1>>	Low Bound	TRUE	TooLow	Permitted	0	900	OFF	OFF	OFF	ON
2	1	RP__1<<1>>	Low Bound	TRUE	TooLow	TooLow	0	0	OFF	OFF	OFF	ON
3	1	RP__1<<2>>	Low Bound	TRUE	Permitted	TooLow	900	0	OFF	OFF	OFF	ON
4	1	RP__1<<2>>	Low Bound	TRUE	Permitted	High	900	1000	OFF	OFF	OFF	ON
5	1	RP__1<<2>>	Low Bound	TRUE	Permitted	Permitted	900	900	OFF	OFF	OFF	ON
6	1	RP__1<<1>>	Hi Bound	TRUE	TooLow	Permitted	899	999	OFF	OFF	OFF	ON
7	1	RP__1<<1>>	Hi Bound	TRUE	TooLow	TooLow	899	899	OFF	OFF	OFF	ON
8	1	RP__1<<2>>	Hi Bound	TRUE	Permitted	TooLow	999	899	OFF	OFF	OFF	ON
9	1	RP__1<<2>>	Hi Bound	TRUE	Permitted	High	999	2000	OFF	OFF	OFF	ON
10	1	RP__1<<2>>	Hi Bound	TRUE	Permitted	Permitted	999	999	OFF	OFF	OFF	ON
11	2	RP__2<<1>>	Low Bound	FALSE	TooLow	Permitted	0	900	OFF	ON	-	-
12	2	RP__2<<1>>	Low Bound	FALSE	TooLow	TooLow	0	0	OFF	ON	-	-
13	2	RP__2<<2>>	Low Bound	FALSE	Permitted	TooLow	900	0	OFF	ON	-	-
14	2	RP__2<<2>>	Low Bound	FALSE	Permitted	High	900	1000	OFF	ON	-	-
15	2	RP__2<<2>>	Low Bound	FALSE	Permitted	Permitted	900	900	OFF	ON	-	-
16	2	RP__2<<3>>	Low Bound	FALSE	Permitted	High	900	1000	OFF	OFF	-	-
17	2	RP__2<<4>>	Low Bound	FALSE	High	Permitted	1000	0	OFF	OFF	-	-
18	2	RP__2<<1>>	Hi Bound	FALSE	TooLow	Permitted	899	999	OFF	ON	-	-
19	2	RP__2<<1>>	Hi Bound	FALSE	TooLow	TooLow	899	899	OFF	ON	-	-
20	2	RP__2<<2>>	Hi Bound	FALSE	Permitted	TooLow	999	899	OFF	ON	-	-
21	2	RP__2<<2>>	Hi Bound	FALSE	Permitted	High	999	2000	OFF	ON	-	-
22	2	RP__2<<2>>	Hi Bound	FALSE	Permitted	Permitted	999	999	OFF	ON	-	-
23	2	RP__2<<3>>	Hi Bound	FALSE	Permitted	High	999	2000	OFF	OFF	-	-
24	2	RP__2<<4>>	Hi Bound	FALSE	High	Permitted	2000	999	OFF	OFF	-	-

6. Acknowledgements

We thank the researchers at NRL, C. Heitmeyer, S. Faulk, B. Labaw, and R. Jeffords for their help on this effort and their comments on drafts of this paper. We also thank P. Ammann and J. Offutt for their continuous guidance in this research.

7. Reference

[BB86] Busser, R. D., M. R. Blackburn, Moving structured methods towards proof of correctness, Proceedings of Structured Development Forum VIII, August 1986.

[BB96] Blackburn, M. R., Busser, R. D., T-VEC: A Tool for Developing Critical System, Eleventh International Conference on Computer Assurance, June 1996.

[Bei83] Beizer, B., Software Testing Techniques, Van Nostrand Reinhold, 1983.

[Bus88] Busser, R. D., Formalizing a theory of real-time software specification, Software Engineering and Its Application to Avionics, NATO Advisory Group for Aerospace Research and Development, April 1988.

[CGR93] Craigen, D., S. Gerhart, T. Ralston, An International Survey of Industrial Applications of Formal Methods, Naval Research Laboratory, NRL/FR/5546—93-9581, Sept. 1993.

[FBWK92] Faulk, S. R., J. Brackett, P. Ward, and J. Kirby, Jr., The Core Method for Real Time Requirements, IEEE Software, Vol. 9, No. 5, September 1992.

[GW94] Ghiassi, M., K. I. S. Woldman, Dual Programming Approach to Software Testing, Software Quality Journal 3, 1994.

[HBGL95] Heitmeyer, C., A. Bull, C. Gasarch and B. Labaw, SCR*: A Toolset for Specifying and Analyzing Requirements, Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95), June 1995.

[Hen80] Heninger, K., Specifying Software Requirements for Complex Systems: New Techniques and Their Application, IEEE Transactions on Software Engineering, Vol. SE6, No. 1, Jan, 1980.

[HJL96] Heitmeyer, C., R. Jeffords, B. Labaw, Automated Consistency Checking of Requirements Specifications, ACM TOSEM, Vol. 5, No. 3, July, 1996.

[HLK95] Heitmeyer, C., B. Labaw and D. Kiskis, Consistency Checking of SCR-Style Requirements Specifications, Proceedings, International Symposium on Requirements Engineering, March, 1995.

[Hoa69] C. A. R. Hoare, An Axiomatic Basis for Computer Programming, Communications of the ACM, Vol. 12, No. 10, October 1969.

[How76] Howden, W.E. Reliability of the path analysis testing strategy, IEEE Transactions on Software Engineering, SE-2(3):208-215, Sept. 1976.

[ORS95] Owre, S., J. Rushby, N. Shankar, Analyzing Tabular and State-Transition Specifications in PVS, SRI International, Tech. Report CSL-95-12, June, 1995.

[PM91] Parnas, D., Madley, J. Functional Decomposition for Computer Systems Engineering (Version 2), TR CRL 237, Telecommunication Research Inst. Of Ontario, McMaster Univ. 1991.

[ROT89] Richardson, D. J., O. O'Malley, C. Tittle, Approaches to specification-based testing, ACM SIGSoft 89: Third Symposium on Software Testing, Analysis and Verification, December 1989.

[RTCA92] RTCA/DO178B, Software Considerations in Airborne Systems and Equipment Certification, Requirements and Technical Concepts for Aviation, Washington, D.C., December 1992. This document is known as EUROCAE ED12B in Europe.

[SA96] Sreemani, T., J. M. Atlee, Feasibility of Model Checking Software Requirements, Eleventh International Conference on Computer Assurance, June 1996.

[Sch90] van Schouwen, A. J., The A-7 requirements model: Re-examination for real-time system and an application for monitoring sytems. TR 90-276, Queen's Univ., Kinston, Ont. 1990.

[SPC93] Software Productivity Consortium. Consortium Requirements Engineering Guidebook, SPC–92060–CMC, version 01.00.09. Herndon, Virginia, 1993.

[WC80] White, L. J., E. I. Cohen, A Domain Strategy for Computer Program Testing, IEEE Transactions on Software Engineering, Vol. SE6, No. 3, May 1980.

[Zei89] Zeil, S. J., Perturbation techniques for detecting domain errors, IEEE Transactions on Software Engineering, (15)6:737-746, June 1989.