Automated Structural Testing of Simulink/TargetLink Models via Search-Based Testing assisted by Prior-Search Static Analysis

Benjamin Wilmes
Berlin Institute of Technology
Daimler Center for Automotive IT Innovations (DCAITI)
Berlin, Germany
E-Mail: benjamin.wilmes@dcaiti.com

***Abstract*—Considering the advantages of early testing and the importance of an efficient quality assurance process, the automation of testing software models would be of great benefit, in particular to the automotive industry. Search-based testing has been applied to automate testing of Simulink models. Despite promising results however, the approach lacks efficiency. Working toward a robust tool, this paper presents three static-analysis-based techniques for assisting and improving search-based structural testing of TargetLink-compliant Simulink models. An interval analysis of model internal signals is used to identify unsatisfiable coverage goals and exclude them from the search. A further analysis determines which model inputs a coverage goal actually depends on in order to reduce the search space. We also propose a technique that sequences coverage-goal-related search processes in order to maximize collateral coverage and reduce test suite size. These additional techniques make the search-based approach applied to Simulink more efficient, as first experiments indicate.**

Keywords-Search-Based Testing; Static Analysis; Simulink.

I. Introduction

Today, in many application areas, the creation of embedded controller software relies on model-based design paradigms. Various industries, such as the automotive industry, use Matlab Simulink (SL) [1] as the standard tool to create and simulate dynamic models along with a code generator, for instance TargetLink (TL) [2], in order to automatically derive software code from such models.

As they are normally the first executable artifacts within software development processes, SL models play an important role in testing theory. Industrial testing practice however, usually focuses on higher-level development artifacts, like testing integrated software or systems as a whole. This discrepancy has both traditional and practical reasons. On one hand, current testing processes still require further adaptation to the model-based paradigm. On the other hand, companies are pressed for time in product development and must deal with an increasing demand for innovation. This can lead to a disregard for low-level tests and model tests in particular. Yet focusing too one-sided on tests of higher-level software or system artifacts poses the risk that faults may be found late in the process, which can lead to increased costs, that some faults can hardly be discovered on higher levels, or that certain functionality is not tested at all.

Thus, automating the testing of software models is highly desirable in industrial practice, particularly with regard to what is normally the most time-consuming testing activity: the selection of adequate test cases in the form of model input values (test data). Search-based testing is a dynamic approach to automating this task. It transforms the test data finding problem into an optimization problem and utilizes meta-heuristic search techniques like evolutionary algorithms to solve it. Search-based testing [3] has been studied widely in the past and has also been applied successfully for testing industrial-sized software systems [4]. Both structural (white-box) and functional (black-box) testing can be automated with the search-based approach.

Zhan and Clark [5], as well as Windisch [6], applied search-based testing to structural test data generation for SL models. The work of Windisch not only supports Stateflow (SF) diagrams (which are used fairly often in SL models), but also makes use of an advanced signal generation approach in order to generate realistic test data. While his approach has led to promising results in general, outperforming commercial tools, it lacks efficiency when applied to larger models. Furthermore, it shows difficulties targeting Boolean states and tackling complex dependencies within models [7].

The work presented in this paper is a first step toward overcoming some of these shortcomings by exploiting static model analysis techniques before the search process actually starts. Our scope is test data generation for TL-compliant Simulink models. We aim to improve both the efficiency and effectiveness of the approach by Windisch.

This paper is structured as follows: Section II introduces search-based testing and its application to structural testing of SL models. Section III presents our approach to supporting the search-based technique by integrating three additional analysis techniques. We present a signal range analysis (Section III-A), which captures range information of internal model signals, and in this way, allows partial detection of unreachable model states. We then propose a signal dependency analysis for the purpose of search space reduction (Section III-B). Our third contribution is a sequencing approach, which derives an order in which coverage goals of a structural test are processed by the search (Section III-C). Insight into our implementation along with first experimental results is provided in Section IV, followed by our conclusions in Section V.

II. Background

A. Search-Based Structural Testing

Search-based testing [3] and its application to industrial cases has been extensively studied in the last decade, motivated by its general scalability, in contrast to purely static techniques like symbolic execution.

The general idea of search-based testing is pretty simple: a test data finding problem (which surely differs in its nature depending on the kind of testing) is transformed into an optimization problem by defining a cost function, called a fitness function. This function rates any test data generated by the deployed search algorithm - usually based on information gained from executing the test object with it. The rating must express in as much detail as possible, how far the test data is from being the desired test data. An iteratively working search algorithm uses these fitness ratings to distinguish good test data from bad, and based on this, generates new test data in each iterative cycle. This fully automated procedure continues until test data satisfying the search goal(s) has been found, that is, if a fitness rating has reached a certain threshold or until a predefined number of algorithm iterations have been performed. Various search algorithms have been used in the past. Due to their strength in dealing with diverse search spaces, evolutionary algorithms, like genetic algorithms, were often preferred [8].

Applied to functional testing, the search-based approach is generally utilized to search for violations of a requirement. In this case, a sophisticated fitness function needs to be designed manually when following the standard approach. However, when applied to structural test data generation, fitness functions can be derived completely automatically from the inner structure of the program to be tested.

Structural testing is commonly aimed at deriving test data based on the internal structural elements of the test object, e.g., creating a set of test data which executes all statements of a code function, or all paths in the corresponding control flow graph. Industrial standards like ISO 26262 even demand the consideration of coverage metrics when performing low-level tests. Search-based testing can automate this task for various coverage criteria (like branch or condition coverage) by treating each structural element requiring coverage as a separate search goal, called a coverage goal (CG). Each CG is accompanied by a specific fitness function. Wegener et al. [9] recommend composing the fitness function of the following two metrics: approach level (positive integer value) and branch distance (real value from 0 to 1). Given a test data's execution path in the control flow graph of the test object's code, the approach level describes the smallest number of branch nodes between the structural element to be covered and any covered path element. To create a more detailed and differing rating of generated test cases, the branch distance reflects how far the test object's execution has been from taking the opposite decision at the covered branch node, which is the closest to the structural element to be covered. This approach is suitable for structural testing of program code, like C or Java code.

B. Application to Dynamic Systems

As model-based development is now established in the automotive industry and practitioners have noticed opportunities to test earlier, Windisch [6] as well as Zhan and Clark [5] have transferred the idea of search-based structural testing from code to model level. For SL models, structural coverage criteria similar to the ones known from code testing exist and are commonly accepted in practice. Before addressing the challenges of applying search-based test data generation to SL models, we give a brief introduction to SL. SL is a graphical data-flow language for specifying the behavior of dynamic systems. Syntactically, a SL model consists of functional blocks and lines connecting them, while most of the blocks are equipped with one or more input ports as well as output ports. The semantics of such a model results from the composed functionalities of the involved block types, e.g., sum blocks, relational blocks or delay functions. In addition, event-driven or state-based functionalities can be realized within SL models using SF blocks. A SF block contains an editable Statechart-like automaton.

When applying search-based structural testing to SL models, two fundamental differences compared to its application on code level arise. First, SL models describe time and state dependent processes. Inputs and outputs of SL models, as well as block-connecting lines, are in fact signals. In order to enable reaching all system states, an execution with input sequences (signals) instead of single input values is required. Such complex test data can only be generated with common search algorithms by compressing the data structure, as done by Windisch [10]. His segment-based signal generation approach also considers the necessity for being able to specify the test data signals to be generated (e.g., amplitude bounds and signal nature like wave or impulse form). Second, the aforementioned fitness function approach cannot be fully adopted since SL models are data flow-oriented. There are no execution paths because the execution of a SL model involves the execution of every included block. Hence, a CG-related fitness function addresses only distances to the desired values of one or more model internal signals. For CGs in SF diagrams however, a bipartite fitness approach is possible [7].

Figure 1 visualizes the overall work flow of applying search-based test data generation to structural testing of SL models as described.

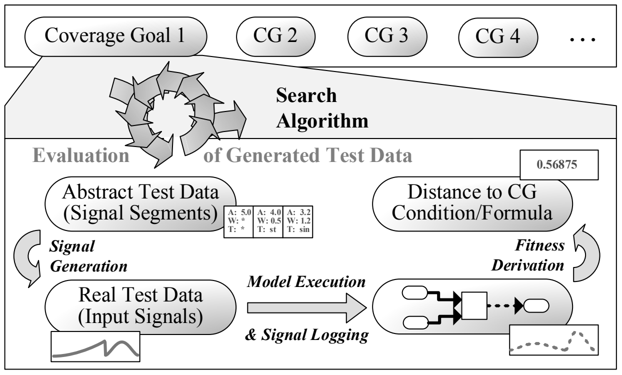
Coverage Goal 1
CG 2
CG 3
CG 4
...
Search Algorithm
Evaluation of Generated Test Data
0.56875
Abstract Test Data (Signal Segments)
Distance to CG Condition/Formula
Signal Generation
Fitness Derivation
Real Test Data (Input Signals)
Model Execution & Signal Logging

Figure 1. Automated search for test sequences, which fulfill coverage goals derived from the model under test

C. Deficiencies and Potential

Search-based structural testing has been applied successfully to real (proprietary) SL models originating from development projects at Daimler, e.g., a model of a windscreen wiper controller [7]. Compared to purely random test data generation of similar complexity, the search-based approach results in significantly higher model coverage. Even in comparison with a commercial tool, the search-based approach performs more effectively.

Despite promising results, the approach lacks efficiency. In general, the overall runtime of the search processes for achieving maximal model coverage increases with the size of the model under test. Similar experiences have been made with code-level search-based structural testing [11]. Since a single automotive SL model is often hundreds of blocks in size, and because a test data generation process of more than a couple of hours is undesirable, improving efficiency of the search-based approach is vital. Apart from the shortcomings of this approach that will be addressed by the contributions in the following sections, there are two other technical problems leading to a lack of efficiency. First, the structural test data generation is performed black-box-like, which means that the model is fed with input values on one end while some distances for calculating fitness are measured at some other point in the model. Any structural information between is not considered, thus the search might be blind to complicated dependencies in the model (cf. [12]). Second, when targeting a Boolean state in a model, a suitable fitness function is hard to find since a simple true or false rating inadequately leads a search [7]. Zhan and Clark suggest a technique called tracing and deducing [13], which mitigates this problem in certain cases, but fails in instances where the Boolean problem cannot be traced back in the model to a non-Boolean one. Further work will address these problems.

As a whole, we aim to improve the search-based approach for structural testing of SL models so that it performs acceptably and reliably in industrial development environments. To this end, we turn our attention to testing of TL-compliant SL models since the code generator TL is widely used in industrial practice. TL extends SL by offering additional block types, but also makes restrictions on the usage of certain SL constructs like block types. Nevertheless, it is possible to adapt our ideas to pure SL usage.

III. Prior-Search Static Analysis

We distinguish between techniques which support search-based structural testing (a) before the CG-related search processes, (b) between the different search processes, (c) during each search process, and (d) after the search processes are done. In the following sections, three techniques belonging to category (a) are presented. Apart from making use of an input specification and choice of coverage criteria provided by the user, all three techniques are fully automatic.

A. Signal Interval Analysis

In structural testing practice, achieving 100% coverage is often not possible. One reason lies in the semantic constructions precluding certain states or signal values. It might also be that a tester specified the test data to be generated in such a way that it prevents certain CGs from being satisfiable. Also, SL models might be designed variably, e.g., contain a constant block with a variable value. When such variability is bound during execution, e.g., via configuration file, certain model states may be unreachable.

CGs referring to unreachable states worsen the overall runtime and undermine the efficiency of search-based structural test data generation since time-consuming search processes are carried out without any hope of finding desired test data. Therefore, we propose two techniques contributing to automatic identification of unreachable CGs. The first one is an interval analysis, which determines the range within which the values of every internal model signal are. If a signal range is in conflict with the range or value required by a CG, this CG is unsatisfiable. We use interval analysis since other approaches to detect infeasibility, such as constraint solving or theorem proving [14], are currently not scalable enough for the complex equations constituted by industrial-sized SL models. The second technique is an analysis of dependencies between CGs. Since this technique is mainly used for another purpose, it is presented in Section III-C.

The code generator TL, as well as the latest version of SL, are capable of analyzing signal ranges in order to perform code optimizations and improve scaling or data type selection, respectively. While those range analysis features are limited (e.g., determining ranges of signals that are involved in loops is not possible without user interaction) our signal interval analysis (SIA) makes use of an input signal specification in order to overcome such limitations and derive more precise ranges.

As mentioned in Section II-B, a tester who uses the search-based approach for testing SL models, as outlined by

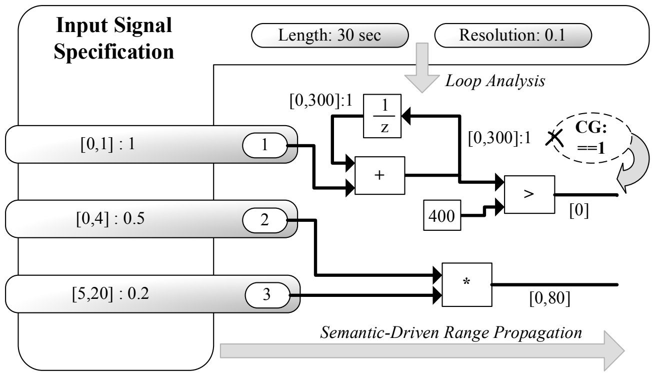

Figure 2. Example of how determining the ranges of a model's internal signals based on input specification and block semantics works

Windisch, is asked to specify the test data to be generated first. This involves establishing (a) the range boundaries and step size of each model input as well as defining (b) a common length (in seconds) and sample rate for all input signals. SIA starts with information (a) for a model's input signals and propagates the corresponding signal ranges of the form $[x, y] : q$, where q (optional) is the step size, through the whole model. Following Wang et al. [15] we also use interval sets instead of a single interval per signal in order to derive more accurate range information. Each propagation step is based on the semantics of the block connecting the signals. In this context, we derived interval semantics for each block type of TL-compliant SL models using basic concepts of interval arithmetic [16].

Figure 2 graphically depicts this procedure with the aid of a simple example. Note that the model contains a loop, initiated by a delay block with an initial value of 0. The standard propagation procedure would be unable to continue here since ranges are not available for all incoming signals of the sum block. A simple, yet imprecise solution is to set the range of the sum block's outgoing signal to the minimum and maximum values of the signal's data type. A more precise solution however, is to use the information (b) of the signal specification in order to run a loop analysis. From length and sample rate, the number of loop iterations is derivable. Starting with the initial value of the delay block a static analysis of the loop iterations is performed, resulting in time-related range information. In order to keep the final results clean and minimal, each signal's ranges as well as the time phases of ranges are combined, if possible, in each iteration of the loop analysis.

Using range propagation and loop analysis in combination, SIA is capable of determining the ranges of all signals contained in the model under test. In cases of blocks with unknown semantics or unsupported blocks, the minimum and maximum values of the outgoing signal's data type are used. Finally, the results of SIA are used to assess whether each CG's associated formula is unsatisfiable - see the exemplary CG in Figure 2. In addition to unsatisfiable CGs, SIA can also identify Boolean signals or discrete signals with only a few possible different values. As described in Section II-C, CGs related to such signals could be problematic for the search-based approach.

B. Signal Dependency Analysis

By default, the search algorithm generates test data for all of the model's inputs when targeting a CG. However, there are usually CGs whose satisfaction is, in fact, independent of the stimulation of certain model inputs. By not taking this into account, the search space is unnecessarily large, which makes it more difficult for the search to find desired test data. To raise efficiency, we include a signal dependency analysis (SDA) to identify which model inputs each CG actually depends on. McMinn et al. [17] investigated a related approach, however, on code level. SDA is closely related to a slicing approach for SL models developed in parallel to our work [18].

At code level, such analysis is usually done by capturing the control dependence in a graph. SL models though, as pointed out previously, are dominated by data dependencies. We therefore analyze the dependency of CGs on input signals by creating a signal dependency graph (a) based on the syntax of the model and (b) refined according to the semantics of blocks which have multiple outgoing signals. Focusing purely on syntax, the following principle leads to a graph describing which signal b the value of a model internal signal a depends on: Signal a is dominated by a signal b if signal a is the outcome of a model block which has signal b incoming. Some blocks with multiple outgoing signals however, do not use every incoming signal in order to calculate the value of a certain outgoing signal. In such cases, the signal dependency graph is refined by removing over-approximated dependencies.

In order to determine which model inputs a certain CG depends on, the signal or signals, which the CG expression refers to, are selected in the dependency graph first. By traversing the graph up to the input signals, the set of relevant model inputs is collected. Within the subsequent search process for this CG, signals are generated only for the relevant inputs - all other inputs receive a standard (e.g., zero valued) signal when being executed.

C. Coverage Goal Sequencing

No matter if structural testing is performed in addition to functional (black-box) testing or purely as white-box testing, it is usually a set of CGs that constitutes the test objective. Remember, that for each CG a separate search needs to be run. In Windisch's approach, those search processes are executed in random order. Hence, correlations between CGs are ignored. Given CGs with the expressions $s<90$, $s<80$ and $s<70$, for example, it is most likely more efficient to aim for reaching the goal $s<70$ first because it satisfies all

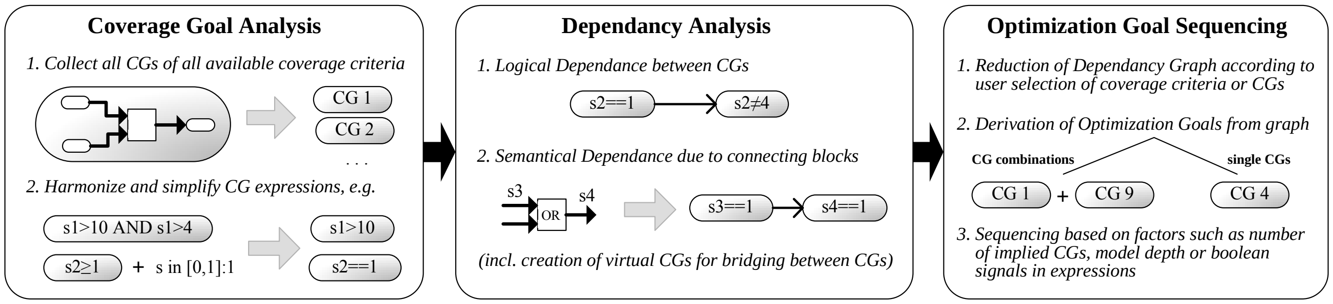
Coverage Goal Analysis
1. Collect all CGs of all available coverage criteria
CG 1
CG 2
. . .
2. Harmonize and simplify CG expressions, e.g.
s1>10 AND s1>4
s1>10
s2≥1
+ s in [0,1]:1
s2==1
Dependancy Analysis
1. Logical Dependance between CGs
s2==1
s2≠4
2. Semantical Dependance due to connecting blocks
s3
s4
OR
s3==1
s4==1
(incl. creation of virtual CGs for bridging between CGs)
Optimization Goal Sequencing
1. Reduction of Dependancy Graph according to user selection of coverage criteria or CGs
2. Derivation of Optimization Goals from graph
CG combinations
single CGs
CG 1
+
CG 9
CG 4
3. Sequencing based on factors such as number of implied CGs, model depth or boolean signals in expressions

Figure 3. Main process steps to create an efficient order for a set of coverage goals, which are processed separately by a search

other CGs at the same time. As this example indicates, the execution order of the CG-related search processes affects the efficiency of the whole structural test.

Other researchers in the search-based testing community have noticed this shortcoming as well. Fraser and Arcuri [19] advise focusing on the generation of whole test suites rather than targeting single CGs. They recommend optimizing multiple test suites instead of multiple test data, and also suggest rewarding smaller test suites with a better fitness in case two or more test suites achieve the same coverage. Harman et al. [20], in contrast, suggest a multi-objective search in which each CG is still targeted individually but the number of collateral (accidentally covered) goals is included as a secondary objective. Though facing a similar problem, our approach differs. We keep the focus on single CGs since, considering the complexity of the optimization problems constituted by industrial SL models, they are often difficult to reach and we do not want to impede the search by burdening it with additional goals or mixed fitness values. Instead, we propose a coverage goal sequencing approach that creates a reasonable order in which the various CGs are pursued. Li et al. worked out a related approach [21], however it is outside of the search-based and SL context. Ultimately, by maximizing collateral coverage, our approach attempts to minimize the number of CGs that need to be pursued. Not only is this expected to improve overall efficiency, but the resulting test suite should also be smaller.

In this paper, we can only give a brief introduction to this technically complex solution, as summarized in Figure 3. First of all, the model under test is analyzed and CGs are derived for all SL/SF-relevant coverage criteria (see [7]). In preparation to analyzing dependencies between CGs, we apply several harmonization and simplification steps to the CG expressions. Note that results of SIA (Section III-A) are used for this task as well, e.g., an expression $s{\geq}1$ would be transformed to $s{=}1$ if s is a Boolean signal. Next, possible dependencies between CG expressions are analyzed, resulting in a set of dependency graphs. A set, since only certain dependencies are considered in order to limit complexity. The following CG relations are registered: implication, equivalence, NAND, and XOR. Dependencies of the expressions are analyzed both from a logical and a semantical point of view. Semantical means that for each two CGs relating to incoming or outgoing signals of the same model block, a block-specific analysis checks if a relation between the CGs exists. In certain cases, the block-specific analysis adds virtual CGs as a bridge to other CGs in order to detect further dependencies. Along the way, based on the captured dependencies, further CGs might be detected as unsatisfiable (see Section III-A).

Considering the user's selection of coverage criteria or single CGs, the graphs are minimized accordingly. Amongst others, non-selected CGs implying selected CGs are kept. From the graphs the final optimization goals are derived in a two-fold way: Besides keeping each selected CG as a single optimization goal, certain combinations of CGs are derived as well - since a graph can contain conjunctions. In this way, optimization goals with high collateral coverage are added. Finally, the optimization goals are sequenced according to several metrics, primarily by the number of (so far unsatisfied) implied CGs, but also by their depth in the model and the amount of Boolean signals involved in the expressions - since such goals should be avoided given the fitness function construction problem (see Section II-C). Note that the pursuing order of optimization goals is updated after each search process ends, since an optimization goal's number of unsatisfied implied CGs might have changed.

IV. Implementation And First Experiments

The presented analysis techniques have been implemented in the course of developing our tool TASMO [22], which is implemented in Java and closely integrated with Matlab. TASMO extracts model related information from Matlab and applies transformation and reduction steps to an internal representation of the model under test in order to focus on the relevant parts for structural test data generation. TASMO can also visualize the results of the presented analysis techniques. We investigated the effect of these techniques to structural test data generation for industrial SL/TL models. Case studies applying the whole procedure, including running the search procedure, are part of ongoing work. We present first experimental results of applying signal interval analysis and signal dependency analysis to two SL models, one mid-sized (model A, 147 blocks, 323 CGs) and one large-sized (model B, 1047 blocks, 736 CGs), recently developed at Daimler in the scope of an electric vehicle's propulsion strategy. Based on the input specifications, SIA identified 17 (A) and 39 (B) infeasible CGs for which the standard search-based approach would otherwise perform extensive search processes. SDA detected that for each CG, on average, only about 3.1 out of 8 (A) and 17.3 out of 32 (B) model inputs would need to be stimulated in order to match the CG formulas. This shows how this technique can reduce the search space distinctly - without excluding CG-relevant search space areas. The runtime of the additional analysis techniques was only a matter of seconds.

V. Conclusion and Future Work

This paper introduces an approach to improving the performance of search-based testing when applied to structural testing of SL models. Three static techniques extend the standard search-based approach by analyzing the model under test before the search processes for each CG are run. Unsatisfiable CGs are partially identified and excluded from the search. The search space is reduced in such a way that the search focuses solely on relevant model inputs. The separate search processes for each CG are sequenced in order to maximize collateral coverage, minimize test suite size, and shorten the overall search runtime.

First experiments backed up our expectations and extensive case studies will follow in the course of our proceeding tool development. We aim to develop a prototype tool that is applicable in industry. Further work is required to extend the presented techniques with full Stateflow support. Targeting the discussed shortcomings of the search-based approach, our next main step is to work on a hybridization of the search-based algorithm with supporting (static) techniques.

References

[1] The Mathworks, "Matlab Simulink," Last access: 2012-09-16. [Online]. Available: http://www.mathworks.com

[2] dSpace, "Targetlink," Last access: 2012-09-16. [Online]. Available: http://www.dspace.com

[3] P. McMinn, "Search-based software testing: Past, present and future," in *Softw. Testing, Verif. and Valid. Workshops (ICSTW)*, 2011.

[4] B. Wilmes, A. Windisch, and F. Lindlar, "Suchbasierter Test für den industriellen Einsatz," in *4. Symp. Test. im Sys.- und Softw. Life-Cycle*, 2011.

[5] Y. Zhan and J. A. Clark, "A search-based framework for automatic testing of MATLAB/Simulink models," *J. Syst. Softw.*, vol. 81, no. 2, pp. 262–285, Feb. 2008.

[6] A. Windisch, "Search-based testing of complex simulink models containing stateflow diagrams," in *Proc. of the 1st Int. Workshop on Search-Based Soft. Testing*, 2008.

[7] A. Windisch, "Suchbasierter Strukturtest für Simulink Modelle," Ph.D. dissertation, Berlin Institute of Technology, 2011.

[8] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Trans. on Softw. Eng.*, vol. 36, pp. 226–247, 2010.

[9] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Inform. and Softw. Technology*, vol. 43, no. 14, pp. 841–854, 2001.

[10] A. Windisch and N. Al Moubayed, "Signal generation for search-based testing of continuous systems," in *Softw. Testing, Verif. and Valid. Workshops (ICSTW)*, 2009.

[11] T. Vos, A. Baars, F. Lindlar, P. Kruse, A. Windisch, and J. Wegener, "Industrial scaled automated structural testing with the evolutionary testing tool," in *Proc. of the 3rd Int. Conf. on Softw. Testing, Verif. and Valid.*, 2010.

[12] P. McMinn, M. Harman, D. Binkley, and P. Tonella, "The species per path approach to search based test data generation," in *Proc. of the 2006 Int. Symp. on Softw. Testing and Analysis (ISSTA)*, 2006, pp. 13–24.

[13] Y. Zhan and J. A. Clark, "The state problem for test generation in simulink," in *Proc. of the 8th Annual Conf. on Genetic and Evolutionary Computation*, 2006, pp. 1941–1948.

[14] A. Goldberg, T. C. Wang, and D. Zimmerman, "Applications of feasible path analysis to program testing," in *Proc. of the Int. Symp. on Softw. Testing and Analysis*, 1994, pp. 80–94.

[15] Y. Wang, Y. Gong, J. Chen, Q. Xiao, and Z. Yang, "An application of interval analysis in software static analysis," in *IEEE/IFIP Int. Conf. on Embedded and Ubiquitous Computing*, vol. 2, 2008, pp. 367 –372.

[16] R. E. Moore, *Interval Analysis*. Prentice-Hall, 1966.

[17] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener, "Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation," *IEEE Trans. on Softw. Eng.*, vol. 38, pp. 453–477, 2012.

[18] R. Reicherdt and S. Glesner, "Slicing Matlab Simulink models," in *34th Int. Conf. on Softw. Eng.*, 2012, pp. 551 –561.

[19] G. Fraser and A. Arcuri, "Evolutionary generation of whole test suites," in *11th Int. Conf. on Quality Softw.*, 2011.

[20] M. Harman, S. G. Kim, K. Lakhotia, P. McMinn, and S. Yoo, "Optimizing for the number of tests generated in search based test data generation with an application to the oracle cost problem," in *Softw. Testing, Verif. and Valid. Workshops (ICSTW)*, 2010, pp. 182–191.

[21] J. J. Li, D. Weiss, and H. Yee, "Code-coverage guided prioritized test generation," *Inf. Softw. Technol.*, vol. 48, no. 12, pp. 1187–1198, 2006.

[22] B. Wilmes, "Toward a tool for search-based testing of Simulink/TargetLink models," in *4th Symp. on Search Based Softw. Eng. (Fast Abstracts)*, 2012.