# Lattice-Based Semantics for Combinatorial Model Evolution

Rachel Tzoref-Brill[1,2(✉)] and Shahar Maoz[1]

[1] School of Computer Science, Tel Aviv University, Tel Aviv, Israel
maoz@cs.tau.ac.il
[2] IBM, Haifa Research Lab, Haifa, Israel
rachelt@il.ibm.com

**Abstract.** Combinatorial test design (CTD) is an effective test design technique, considered to be a testing best practice. CTD provides automatic test plan generation, but it requires a manual definition of the test space in the form of a combinatorial model. As the system under test evolves, e.g., due to iterative development processes and bug fixing, so does the test space, and thus, in the context of CTD, evolution translates into frequent manual model definition updates.

In this work, we show that the Boolean semantics currently in use by CTD tools to interpret the model is inadequate for combinatorial model evolution, and propose to replace it with a new lattice-based semantics that (1) provides a consistent interpretation of atomic changes to the model via Galois connections, and (2) exposes which additional parts of the model must change following an atomic change, in order to restore validity. We further use the new lattice-based semantics to define new higher-level atomic operations for combinatorial model evolution. Finally, we identify recurring abstraction and refinement patterns in the evolution of 42 real-world industrial models, and use the new lattice-based semantics to define new higher-level atomic constructs that encapsulate these patterns.

The proposed lattice-based semantics and related new modeling constructs advance the state-of-the-art in CTD with a new foundation for model evolution and with better tools for change comprehension and management.

## 1 Introduction

Combinatorial test design (CTD) is an effective technique for coping with the verification challenge of increasingly complex software systems, and is considered a testing best practice [1,2,5,8,26]. In CTD, the test space is manually modeled

This research was done under the terms of a joint study agreement between IBM Corporation Inc (via the IBM Research Lab - Haifa) and Tel Aviv University. Additionally, part of the research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement no. 610802.


B. Finkbeiner et al. (Eds.): ATVA 2015, LNCS 9364, pp. 276–292, 2015.
DOI: 10.1007/978-3-319-24953-7_22

by a set of parameters, their respective values, and constraints on the value combinations. The aggregate of parameters, values, and constraints is called a *combinatorial model*. We refer to the parameters and their values as the *model domain*. A valid test in the test space is defined to be an assignment of one value to each parameter without violating the constraints. A subset of the space is automatically constructed so that it covers all valid value combinations of every $t$ parameters, where $t$ is usually a user input. This systematic selection of tests is based on empirical data that shows that in most cases, the appearance of a bug depends on the interaction between a small number of features of the system under test [5,12,24].

An under-explored challenge for wide deployment of CTD in industry is the manual process for modeling and maintaining the test space. A recent survey by Nie et al. [16] reveals that only around 5 % of the publications on CTD explore the crucial modeling process. The topic of model maintenance is not even mentioned in [16]. However, in practice, model maintenance is of significant importance, since creating a CTD model is not a one time effort. Models must be maintained during the evolution of the system under test, and this need is significantly strengthened by the move to agile methodology and to continuous delivery mode. As software development cycles are getting ever shorter, with monthly and weekly deliveries, test design needs to frequently adjust to changes, which in the context of CTD means frequent model definition updates. While in these settings technologies for handling model changes are increasingly necessary, we are unaware of any work that reasons about the evolution process of combinatorial models or provides tool support for it. Close to 40 CTD tools are listed in [18], for example PICT [4] and ACTS [21]. However, to the best of our knowledge, none of the existing tools provides indication on the effect of change operations on the model, i.e., what is the relation between the original model and the new one, and how they differ. The practitioner is "left in the dark" as to whether the performed change will result in the intended effect, and what other changes are required. This problem exacerbates when a series of such change operations is performed, as is typically the case.

To start examining the evolution of combinatorial models in practice, as a preliminary study, we have gathered information about the evolution of 42 real-world industrial combinatorial models, which have been created over the last 7 years, each with 2–5 versions.[1] The number of parameters in a model version ranged from 4 to 109 (arithmetic mean 18.3); the number of constraints ranged from 0 to 381 (arithmetic mean 32.2). A total of 107 artifacts and 65 version commits were examined. The models originate from various domains and were written by different CTD practitioners. When analyzing the semantic changes applied to the models, we observed 6 recurring patterns, 4 of which are abstraction and refinement patterns. The frequency of occurrence of the latter patterns in real-world model evolution motivated us to try and interpret the evolution

[1] Unfortunately, all models are confidential and cannot be shared since they were created for IBM or for its clients. We are in the process of checking the option of sharing most of them after obfuscation.

process using abstraction and refinement techniques. In fact, abstraction and/or refinement patterns were observed in all models, and in 90 % of commits. The first two patterns we identified are *contraction* (in 38 % of the commits) and *extension* (in 52 % of the commits), where parts are either removed from or added to the model to describe the test space with less or more detail. The next two abstraction and refinement patterns we identified are merge and split. These patterns replace existing parts of the model with new parts that describe the original parts with less or more detail. Neither merge nor split are currently directly supported in existing CTD tools, though we detected split in 26 % of the version commits and merge in 14 % of them, implicitly implemented as a series of atomic operations. The last two change pattens we observed were refactoring and correction, both observed in 26 % of the commits.

In this work, we explore the evolution process of combinatorial test space modeling. We show that the Boolean semantics currently in use by CTD tools is inconsistent and inadequate for combinatorial model evolution, and extend it to a lattice-based semantics that provides a consistent interpretation of model changes via Galois connections. A Galois connection establishes the connection between the elements in an abstract domain and those in a concrete domain, via abstraction and concretization functions. We use these functions to uniquely define the validity of tests when moving from a certain test space domain to a more concrete or more abstract domain via an operation on the model domain. The reasoning behind the abstraction and refinement interpretation is that in the inherently incremental and forward fashion evolution process, each change is interpreted in isolation, without knowing the larger context and what changes will follow. In such settings, abstraction and refinement are natural interpretations of model domain changes, since each such change either hides details from the model or adds details to it.

We first address atomic change operations, e.g., adding or removing a parameter and its values, and use the new semantics to present their formal interpretation as either abstractions or refinements. We also use the new semantics as a means to mark the tests whose validity is defined as unknown following such changes in the model, and thus expose the fact that some changes in the model require additional changes to follow.

We further propose two higher-level change operations, which we call merge and split, that capture the merge and split change patterns observed in real-world model evolution. We use our lattice-based semantics to extend the CTD modeling language with constructs that encapsulate these change patterns and enable their safe and consistent application.

It is important to note that in this work we focus on the evolution at the model level, and do not address the evolution of the test plans generated from the model. Furthermore, our work is completely independent of the criteria and algorithm used to generate test plans. Reasoning about co-evolution of models and the test plans derived from them is part of our future work plans.

We implemented the new semantics and related constructs in our industrial-strength commercial CTD tool IBM Functional Coverage Unified Solution (IBM FOCUS) [10,22]. For scalability, the implementation is symbolic and is based on

**Table 1.** Example on-line shopping model

| Parameter | Values |
|---|---|
| ItemStatus (IS) | InStock, OutOfStock, NoSuchProduct |
| OrderShipping (OS) | Air, Ground |
| DeliveryTimeframe (DT) | Immediate, OneWeek, OneMonth |

| Constraints |
|---|
| $\texttt{DT} = \texttt{Immediate} \rightarrow \texttt{OS} = \texttt{Air}$ |
| $\texttt{DT} = \texttt{OneMonth} \rightarrow \texttt{OS} = \texttt{Ground}$ |

Binary Decision Diagrams. The proposed semantics and related new modeling constructs advance the state-of-the-art in CTD with a new foundation for model evolution and with better tools for change comprehension and management.

## 2 Running Example and Overview

We start off with an example and overview of our work. The presentation in this section is semi-formal. Formal definitions appear in Sects. 3 to 6.

Table 1 depicts the parameters, values, and constraints of a combinatorial model for an on-line shopping system, which we use as a running example.

The model defines the test space and which tests in it are valid. For example, the test $(\texttt{IS} = \texttt{InStock}, \texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{Immediate})$ is valid, while the test $(\texttt{IS} = \texttt{InStock}, \texttt{OS} = \texttt{Ground}, \texttt{DT} = \texttt{Immediate})$ is invalid.

**Example 1: Adding a Value to an Existing Parameter.** Following the addition of a new feature to the system, a practitioner added the value `Sea` to the parameter `OrderShipping`. Using the Boolean semantics, as is the case in all existing CTD tools, we get that the question whether the test $(\texttt{IS} = \texttt{InStock}, \texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{Immediate})$ is valid is inconsistently answered, depending on the syntactic representation of the test space. If the constraints are written as in Table 1, then this test is invalid. In contrast, if the first constraint was instead written $\texttt{DT} = \texttt{Immediate} \rightarrow \texttt{OS} \neq \texttt{Ground}$, then the same test is valid. Thus, although the two original models have equal semantics (they induce the same set of valid tests), adding the value `Sea` to the parameter `OS` results in two different new models. Moreover, the practitioner is not informed of this inconsistency and of what further input is required to resolve it. As a result, tests in the new model might not be assigned with their intended validity.

Our work addresses this problem. With our new, lattice-based semantics, the test in question is assigned with an "unknown" validity, regardless of the syntax in which the original constraints were specified. Intuitively, its validity depends on the order shipping value, for which the practitioner has yet to provide validity information, and is thus defined as unknown. In contrast, the test $(\texttt{IS} = \texttt{InStock}, \texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{OneWeek})$ will be valid in our semantics, because

its validity is determined regardless of the order shipping value. Moreover, our tool identifies and marks the tests that have an unknown validity following the addition of the `Sea` value, namely all tests that contain the combinations $(\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{Immediate})$ or $(\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{OneMonth})$, and presents them in a concise form to the practitioner for further validity specification. Thus, our semantics enables exposing to the practitioner what additional changes are required following the addition of the value to the model.

**Example 2: Splitting a Value.** After further inquiries, the practitioner realized that delivery time frame of one month actually consists of two different values, `6To10WorkingDays` and `Over10WorkingDays`, which represent two separate logical paths of the application under test. To change the model accordingly using existing CTD tools, the practitioner needs to remove the `OneMonth` value, add the two new values, and change the constraints to consider the split values, without any indication from the tool regarding the effect of each step and the required consequent steps.

We call this type of change a split. Using our semantics, a split is formally presented as a refinement of the model domain, where the split values refine the original value. According to our semantics, a test containing a `6To10WorkingDays` or `Over10WorkingDays` value automatically inherits the validity of the same test with the `OneMonth` value, which is no longer in the model. Our tool offers a *split* operation that incorporates the above three steps, based on our semantics.

Note that in all existing tools, the practitioner can perform the split only as a sequence of separate steps; after the first two steps, the validity of the different tests in Boolean semantics depends on the syntax in which the original constraints were written, and might differ from the same tests containing the original value. Following the removal of `OneMonth` and the addition of `6To10WorkingDays` and `Over10WorkingDays`, the set of valid tests will contain the pair $(\texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{6To10WorkingDays})$, while it will not contain the pair $(\texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{OneMonth})$. However, this new pair could also be excluded from the set of valid tests if the syntax of the original constraints were different, e.g., $\texttt{DT} \neq \texttt{Immediate} \wedge \texttt{DT} \neq \texttt{OneWeek} \rightarrow \texttt{OS} = \texttt{Ground}$. Furthermore, as in the case of adding the `Sea` value, there would be no indication that the constraints should change in order to be consistent with the original model, and thus the practitioner might neglect changing them. The required changes may be easy to manually identify when there are only two constraints in the model, but they are much more challenging when the model contains dozens of parameters and constraints, as is typically the case in real-world industrial models.

**Alternative Solutions.** One may suggest alternative solutions to overcome the inconsistent constraints interpretation following extensions to the model domain, and the incomplete interpretation following removals from the model domain. For example, to handle the former problem, one may suggest to remove the negation operator from the constraint language made available to the practitioner. While this will resolve the inconsistent interpretation, it will extremely limit the flexibility of the practitioner to specify constraints in a concise manner,

and is thus infeasible in practice. To handle the latter problem, one may suggest to use propositional semantics, where each partial parameter assignment maps each constraint to a new constraint. Such a solution will result in numerous versions of the constraints, which the practitioner needs to reason about following each single removal from the model. In contrast to these alternatives, our proposed lattice-based semantics handles both problems at once, without limiting the constraints specification language and without requiring the practitioner to deal with overly complex information.

## 3 Preliminaries

We provide formal definitions for the mathematical constructs that will be used throughout this work, the combinatorial model, which defines a test space, and lattices and Galois connections, which we use as a basis for the new semantics.

**Combinatorial Models.** A combinatorial model is defined as follows. Let $P = \{p_1, \ldots, p_n\}$ be an ordered set of parameters, $V = \{V_1, \ldots, V_n\}$ an ordered set of finite value sets, where $V_i$ is the set of values for $p_i$, and $C$ a set of Boolean propositional constraints over $P$. A test $(v_1, \ldots, v_n)$, where $\forall_i,\ v_i \in V_i$, is a tuple of assignments to the parameters in $P$.

The current semantics used in practice by CTD tools [18] is Boolean semantics. In this semantics, a valid test is a test that satisfies all constraints in $C$. The semantics of the model is the set of all its valid tests, denoted by $S(P, V, C)$.

**Complete Lattices and Galois Connections.** A *lattice* is a tuple $L = \langle D, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$, where $D$ is a set of elements, $\sqsubseteq$ is a partial order on $D$, $\sqcup$ is the join operator that defines a least upper bound for every finite set of elements in $D$, $\sqcap$ is the meet operator that defines a greatest lower bound for every finite set of elements in $D$, $\bot$ is the bottom element in $D$, defined as $\sqcup\emptyset$, and $\top$ is the top element in $D$, defined as $\sqcup D$. $L$ is a *complete lattice* if the meet and join operators are defined for arbitrary sets in $D$, rather than only for finite ones.

Given two complete lattices $C = \langle D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C \rangle$ and $A = \langle D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A \bot^A, \top^A \rangle$, representing the concrete domain and the abstract domain, respectively, a *Galois connection* is a quadruple $(C, \alpha, \gamma, A)$ that relates $C$ and $A$ via a monotone abstraction function $\alpha : D^C \to D^A$ and a monotone concretization function $\gamma : D^A \to D^C$. It must hold that $\forall c \in D^C, c \sqsubseteq \gamma(\alpha(c))$ and $\forall a \in D^A, \alpha(\gamma(a)) \sqsubseteq a$. A *Galois insertion* is a Galois connection for which $\forall a \in D^A,\ \alpha(\gamma(a)) = a$. Specifically in our work, $c$ represents a set of tests in the concrete domain, and $a$ represents a test in the abstract domain. The concrete domain is derived from the abstract domain by an atomic operation such as the addition of a value to a parameter. The abstract domain is derived from the concrete domain by the reverse operation, e.g., the removal of the value from the parameter. $\alpha$ maps $c$ to a test which is its most precise abstraction in the abstract domain, and $\gamma$ maps $a$ to a set of tests which is its most general concretization in the concrete domain. A mathematical introduction to lattices and Galois connections can be found in [7].

Finally, given a set $V$, a complete lattice $L$, and a function $\beta : V \rightarrow L$, termed an *extraction function*, the following quadruple $(2^V, \alpha, \gamma, L)$ is a Galois connection between $2^V$ and $L$, where $\alpha(X) = \sqcup\{\beta(v) \mid v \in X\}$ for any $X \subseteq V$, and $\gamma(a) = \{v \in V \mid \beta(v) \sqsubseteq a\}$ [17].

## 4 Lattice-Based Semantics for Combinatorial Model Evolution

We are now ready to present the main contribution of our work, a lattice-based semantics for combinatorial model evolution. We define the new semantics to be consistent with Boolean semantics for a single version of the model, yet to provide a consistent definition of the model following each atomic change in the domain of the model and to mark the assignments whose validity is unknown following such changes.

Our lattice-based semantics for combinatorial models consists of two lattices: an *information lattice*, which captures the parameter assignments, and is by itself a Cartesian product of lattices per parameters, and a *validity lattice*, which captures the validity information for the parameter assignments.

Formally, let $P = \{p_1, \ldots, p_n\}$ be a set of parameters, and $V = \{V_1, \ldots, V_n\}$ a set of finite value sets, where $V_i = \{v_{i1}, \ldots, v_{im_i}\}$ is the set of values for $p_i$. Let $V_{i*}$ be the set $V_i \bigcup \{\top, \bot\}$. For each parameter $p_i$ we define a complete lattice of possible assignments $L_i = \langle V_{i*}, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$, where $\sqsubseteq$ is a partial order on $V_{i*}$ in which the elements of $V_i$ are not ordered, $\top$ represents the unknown value which contains the least information about the assignment to $p_i$ and therefore is the top element of $L_i$, and $\bot$ represents a contradicting assignment to $p_i$ and is therefore the bottom element of $L_i$.

The information lattice is built as the Cartesian product of the lattices defined above: $L_{cart} = \langle V_{1*} \times V_{2*} \times \ldots \times V_{n*}, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \bot_{cart}, \top_{cart} \rangle$, where $\sqsubseteq_{cart}$ is the point-wise application of $\sqsubseteq$, and $\sqcup_{cart}$ and $\sqcap_{cart}$ are the point-wise application of $\sqcup$ and $\sqcap$, respectively. $\bot_{cart}$ and $\top_{cart}$ are the elements resulting from the assignment of $\bot$ and $\top$ to all parameters in $P$, respectively. For simplicity, we merge all tests that contain a $\bot$ value into the $\bot_{cart}$ element. As will become clear in the sequel, the implications of this are minor, as all of these tests map to the empty set of concrete tests, whether valid or not.

Validity is given using a second complete lattice $L_{valid} = \langle \{0, 1, \bot_{valid}, \top_{valid}\}, \sqsubseteq_{valid}, \sqcup_{valid}, \sqcap_{valid}, \bot_{valid}, \top_{valid} \rangle$, where 0 stands for invalid, 1 stands for valid, $\top_{valid}$ stands for unknown validity, and $\bot_{valid}$ stands for contradicting validity. $\sqsubseteq_{valid}$ is a partial order on $\{0, 1, \bot_{valid}, \top_{valid}\}$ in which 0 and 1 are not ordered. Figure 1 depicts the information lattice of our on-line shopping example as well as the validity lattice.

We further define a validity function $F_{valid} : L_{cart} \rightarrow L_{valid}$ that maps each element in the Cartesian product to its validity information. A test is an element $t \in L_{cart}$. We denote the value of a parameter $p \in P$ in a test $t \in L_{cart}$ by $t(p)$. A test is a complete test if $\forall p \in P, t(p) \neq \top$. Otherwise it is a partial test. The set

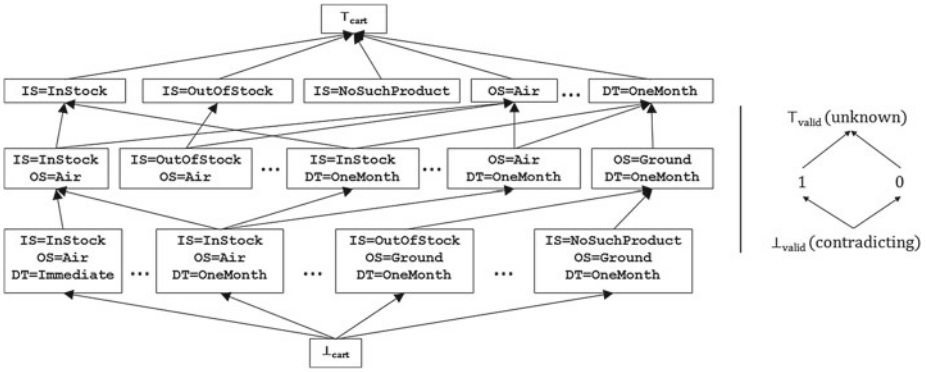


**Fig. 1.** On the left, part of the information lattice $L_{cart}$ of our on-line shopping example. Parameters that do not appear in a square are assigned with the $\top$ (unknown) value. On the right, the validity lattice $L_{valid}$.

of valid tests consists of all complete tests $t \in L_{cart}$ such that $F_{valid}(t) = 1$. The set of invalid tests consists of all complete tests $t \in L_{cart}$ such that $F_{valid}(t) = 0$.

We impose the following requirements on $F_{valid}$, to make it consistent with Boolean propositional constraints, as will be shown by Lemma 2:

1. **Monotonicity.** $\forall t_1, t_2 \in L_{cart}$, if $t_1 \sqsubseteq t_2$ then $F_{valid}(t_1) \sqsubseteq F_{valid}(t_2)$. That is, the more information is given on the parameter assignment, the more information is deduced on the validity of the assignment.
2. **Consistency.** $F_{valid}(t) = \bot_{valid}$ iff $t = \bot_{cart}$.
3. **Completeness.** For a single version of the model, we require that for every complete test $t$, $F_{valid}(t) \neq \top_{valid}$. This will no longer be a requirement during the evolution of the model.

$F_{valid}$ induces two dual properties: *strongest exclusions* and *strongest inclusions*. A test $t \in L_{cart}$ is a strongest exclusion if $F_{valid}(t) = 0$ and $\forall t', t \sqsubset t' \rightarrow F_{valid}(t') = \top_{valid}$. In other words, a strongest exclusion contains the least assignment information needed to deduce its invalidity. A test $t \in L_{cart}$ is a strongest inclusion if $F_{valid}(t) = 1$ and $\forall t', t \sqsubset t' \rightarrow F_{valid}(t') = \top_{valid}$. In other words, a strongest inclusion contains the least assignment information needed to deduce its validity.

These properties are important as they will allow us to describe $F_{valid}$ and the changes that occur to it uniquely and succinctly. The validity of the other tests is directly determined by the validity of the strongest exclusions and strongest inclusions. Going back to our on-line shopping model, Table 2 presents its $F_{valid}$ function via its strongest exclusions and strongest inclusions.

**Lemma 1.** *For a single version of a model, (a) the set of strongest exclusions $T_{ex}$ uniquely determines $F_{valid}$, and (b) the set of strongest inclusions $T_{inc}$ uniquely determines $F_{valid}$.*

The proof relies on the consistency, monotonicity, and completeness requirements. For details see the long version of the paper at [25].

**Table 2.** Strongest inclusions and exclusions for the on-line shopping model

| |
|---|
| $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \top, \texttt{DT} = \texttt{OneWeek}) = 1$ |
| $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Ground}, \texttt{DT} = \texttt{OneMonth}) = 1$ |
| $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{OneMonth}) = 0$ |
| $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{Immediate}) = 1$ |
| $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Ground}, \texttt{DT} = \texttt{Immediate}) = 0$ |

**Lemma 2.** *For a single version of a model, lattice-based semantics is consistent with Boolean semantics.*

To prove Lemma 2, we show that a model in Boolean semantics $S(P, V, C)$ can be represented in lattice-based semantics by creating an $F_{valid}$ function that is consistent with the set of valid tests captured by $S(P, V, C)$. Similarly, given a model in lattice-based semantics $(P_L, V_L, F_{valid})$, we can create a model in Boolean semantics $S(P_L, V_L, C)$ whose set of valid tests is $\{t \in L_{cart} \mid F_{valid}(t) = 1\}$. The full proof appears in [25].

## 5 Atomic Operations on Combinatorial Models

We observe the following atomic operations that are performed by practitioners on combinatorial models: adding a parameter and its values, removing a parameter and its values, adding a value to an existing parameter, removing a value from an existing parameter, and adding, removing, or changing a constraint.

In the following, we provide semantics for atomic operations on combinatorial models. We interpret each atomic change in the domain of the model as either an abstraction or a refinement via a Galois connection. Abstraction and refinement are natural interpretations of an atomic domain change when viewing it in isolation, without knowing what consequent changes will occur. The Galois connection is a means to automatically provide a consistent definition of the validity of the entire model following a domain change, via its abstraction and concretization functions. We interpret an atomic change in the validity function as an abstraction, refinement, refactoring, correction, or as a mixture of these change types. Higher-level semantic changes are discussed in Sect. 6.

### 5.1 Adding or Removing a Parameter and Its Values

Adding a parameter $p_i$ with a set of values $V_{i*} = V_i \cup \{\top, \bot\}$ to the model results in a new domain $L'_{cart}$ defined over the set of parameters $P' = P \cup \{p_i\}$ and values $V' = V \cup V_{i*}$. This domain is a refinement of the original domain $L_{cart}$.

Similarly, removing a parameter $p_i$ and its values $V_{i*}$ from the model results in a new domain which is an abstraction of the original one. The connection between the two domains can be described as a Galois connection $(2^{L'_{cart}}, \alpha, \gamma, L_{cart})$ via an extraction function as follows.

**Definition 1 (Galois connection for adding or removing a parameter).** *Let* $\beta : L'_{cart} \rightarrow L_{cart}$ *be an extraction function, where* $\forall t \in L'_{cart}, \beta(t) = t \setminus p_i$. *Then* $\beta$ *uniquely defines* $\alpha$ *and* $\gamma$ *functions as follows:* $\forall T \subseteq L'_{cart}, \alpha(T) = \sqcup_{cart}\{\beta(t) \mid t \in T\}$, *and* $\forall t \in L_{cart}, \gamma(t) = \cup\{t' \in L'_{cart} \mid \beta(t') \sqsubseteq_{cart} t\}$.

The intuition behind this reasoning is that the additional parameter adds more details to the domain and to each test derived from this domain, and therefore is a form of refinement. Going back to our on-line shopping model from Sect. 2, adding the parameter `ExportControl` with values `true, false`, refines the test space by describing in each test whether or not export control is required. Similarly, removing a parameter hides the information it represents, and is therefore a form of abstraction. For example, removing the parameter `IS` from the model eliminates from each test the information about the status of the item to be purchased.

**Theorem 1.** $(2^{L'_{cart}}, \alpha, \gamma, L_{cart})$ *is a Galois insertion.*

The proof follows directly from the definitions of $\alpha$, $\beta$, and $\gamma$. For details see [25].

Though the addition or removal of a parameter does not directly change the validity function, it implicitly influences it. As demonstrated in Sect. 2, such changes might result in an inconsistent or inadequate interpretation of the model when Boolean semantics is considered. For example, if a removed parameter appears in the constraints, they can no longer be interpreted with Boolean semantics in the new model due to the missing information.

In contrast, the lattice-based semantics provides a consistent and unique definition of the validity function following a change in the model domain, via the Galois connection. Let $F_{valid} : L_{cart} \rightarrow L_{valid}$ be the validity function in the domain without the additional parameter $p_i$, and $F'_{valid} : L'_{cart} \rightarrow L_{valid}$ be the validity function in the domain that includes it. Then the validity functions are defined as follows.

**Definition 2 (Validity function following the addition or removal of a parameter).** $\forall t \in L'_{cart}, F'_{valid}(t) = F_{valid}(\alpha(\{t\})) = F_{valid}(\beta(t))$;$\forall t \in L_{cart}, F_{valid}(t) = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in \gamma(t)\}$.

**Lemma 3.** $F_{valid}$ *and* $F'_{valid}$ *preserve monotonicity and consistency.*

For proof refer to [25].

Note that Definition 2 does not depend on the syntax of the constraints, but rather on Definition 1, i.e., on the Galois connection relating the two domains. According to Definition 2, when a parameter $p_i$ is added to the model, $L'_{cart}$ "inherits" the validity of $L_{cart}$. That is, $\forall (t, p_i = v) \in L'_{cart}, F'_{valid}((t, p_i = v)) = F_{valid}(\beta((t, p_i = v))) = F_{valid}(t)$. This is indeed the extension of $F_{valid}$

to $F'_{valid}$ that CTD tools implicitly use. For example, when the parameter `ExportControl` is added to the on-line shopping model, the validity of all tests is determined regardless of the export control information, although they contain it. Further refinement of the validity information based on the added parameter can be performed by the practitioner in subsequent steps.

When a parameter is removed from the model, existing CTD tools mark the constraints in which it appears as erroneous, and the validity function becomes undefined until the constraints are corrected. For example, if the `OS` parameter is removed from the on-line shopping model, the validity information cannot be computed anymore in Boolean semantics.

In contrast, using Definition 2, we get the following: $\forall t \in L_{cart}, F_{valid}(t) = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in \gamma(t)\} = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in L'_{cart} \wedge \beta(t') \sqsubseteq_{cart} t\} = \sqcup_{valid}\{F'_{valid}((t'', p_i = v)) \mid t'' \sqsubseteq_{cart} t, v \in V_{i*}\} = F'_{valid}(\sqcup_{cart'}\{(t'', p_i = v) \mid t'' \sqsubseteq_{cart} t, v \in V_{i*}\}) = F'_{valid}((t, p_i = \top))$.

That is, the validity of each test in the more abstract model is determined by assigning the removed parameter with the unknown ($\top$) value. If the validity can still be determined, it is derived from the refined model. If it cannot be determined, it becomes unknown ($\top_{valid}$). The advantage of our semantics is that it pinpoints the exact parameter assignments for which the removal of $p_i$ leads to an unknown validity, and therefore may need to be refined. In our example, though `OS` was removed from the model, the validity of the partial test (`IS` $= \top$, `DT` $=$ `OneWeek`) is 1, while the validity of the partial tests (`IS` $= \top$, `DT` $=$ `Immediate`) and (`IS` $= \top$, `DT` $=$ `OneMonth`) is $\top_{valid}$. This again demonstrates how the lattice-based semantics provides the practitioner with more refined information about which parts of the model lost validity information and require further input, and which parts maintained complete validity information following the change and can be kept as is.

### 5.2 Adding or Removing a Value from an Existing Parameter

Adding a value $v_{ik}$ to a parameter $p_i$ results in a new domain $L'_{cart}$ defined over $P$ and the set of values $V^*$, where $V^*(p_i) = V(p_i) \cup \{v_{ik}\}$. This domain is a refinement of the previous domain $L_{cart}$. Similarly, removing a value $v_{ik}$ from a parameter $p_i$ results in a new domain which is an abstraction of the previous one. Note that $L_{cart} \subset L'_{cart}$. The connection between the two domains can be described as a Galois connection $(2^{L'_{cart}}, \alpha', \gamma', L_{cart})$ via an extraction function as follows.

**Definition 3 (Galois connection for adding or removing a value).** *Let $\beta' : L'_{cart} \rightarrow L_{cart}$ be an extraction function, where $\forall t \in L'_{cart}, \beta'(t) = t[v_{ik}/\top]$. Then $\beta'$ uniquely defines $\alpha'$ and $\gamma'$ functions as follows: $\forall T \subseteq L'_{cart}, \alpha'(T) = \sqcup_{cart}\{\beta'(t) \mid t \in T\}$, and $\forall t \in L_{cart}, \gamma'(t) = \cup\{t' \in L'_{cart} \mid \beta'(t') \sqsubseteq_{cart} t\}$.*

Similarly to the previous case, the intuition behind this reasoning is that the additional value adds more details to the domain, this time with respect to the specific parameter it is being added to, and therefore is a form of refinement.

Adding the value `Sea` to the `OS` parameter refines the information about the available shipping methods. Similarly, removing a value from a parameter hides the information it represents for the parameter, and is therefore a form of abstraction. For example, removing the value `NoSuchProduct` from the parameter `IS` hides the information of whether the item requested to be purchased actually exists.

**Theorem 2.** $(2^{L'_{cart}}, \alpha', \gamma', L_{cart})$ *is a Galois insertion.*

The proof follows directly from the definitions of $\alpha'$, $\beta'$, and $\gamma'$. For details see [25].

As in the previous case, though the validity is not explicitly changed by the atomic operation, it is implicitly influenced by it. Again, the Galois connection provides a consistent and unique definition of the validity function, which exposes the exact cases in which the validity becomes unknown due to the domain change, and therefore requires further information from the practitioner.

**Definition 4 (Validity function following the addition or removal of a value).** $\forall t \in L'_{cart}, F'_{valid}(t) = F_{valid}(\alpha'(\{t\})) = F_{valid}(\beta'(t))$; $\forall t \in L_{cart}$, $F_{valid}(t) = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in \gamma'(t)\}$.

Lemma 3 applies to Definition 4 as well. For details see [25].

When a value is added to the model, according to the definition of $\beta'$, we get that if $t(p_i) \neq v_{ik}$ then $F'_{valid}(t) = F_{valid}(t)$. Otherwise $F'_{valid}(t) = F_{valid}(t')$, where $t'(p_i) = \top \wedge \forall j \neq i, t'(p_j) = t(p_j)$.

That is, the new value is treated as unknown when interpreting the validity of an assignment in which it participates. If the validity can be determined regardless of the value of the parameter $p_i$, then it remains determined. Otherwise, it is defined as unknown, making the validity function for this assignment under-specified and forcing the practitioner to explicitly define it. As shown in Sect. 2, using Definition 4 we are able to deduce that the validity of $(\texttt{IS} = \top, \texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{Immediate})$ is $\top_{valid}$, because the validity of $(\texttt{IS} = \top, \texttt{OS} = \top, \texttt{DT} = \texttt{Immediate})$ is also $\top_{valid}$, regardless of the syntax of the original constraints. In contrast, in Boolean semantics, the test is assigned with conflicting 0 and 1 validity in this case, depending on the syntax of the constraints.

When removing a value $v_{ik}$ of a parameter $p_i$ from the model, we get that $F_{valid}(t) = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in \gamma'(t)\} = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in L'_{cart} \wedge \beta'(t') \sqsubseteq_{cart} t\} = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in L'_{cart} \wedge t'[v_{ik}/\top] \sqsubseteq_{cart} t\} = F'_{valid}(\sqcup_{cart'}\{t' \in L'_{cart} \mid t' \sqsubseteq_{cart'} t\}) = F'_{valid}(t)$. In this case, since $L_{cart} \subset L'_{cart}$, we simply get that the validity of $t$ is inherited from the refined model.

### 5.3 Adding, Removing, or Changing a Constraint

As opposed to a change in the domain of the model that implicitly changes its validity function, an addition, removal, or change in a constraint does not implicitly change the domain of the model. Therefore, it does not require guaranteeing

a consistent interpretation of an implicit change. Instead, we reason about the different semantics of changes in the constraints, such as abstraction, refinement, refactoring, and correction, by analyzing their effect on the strongest exclusions and strongest inclusions. Since this analysis is an extension of our proposed semantics that is not required for the understanding of the main contribution of this work, we opt for describing it in [25].

## 6 Split and Merge Operations on Combinatorial Models

We now move from analyzing each atomic operation in isolation, to examining higher-level operations that stand for a semantic change. Specifically, we focus on two such changes which we call *merge* and *split*. These changes replace existing parts of the model with new parts that describe the original parts with less or more detail. We provide formal definitions for split and merge using our lattice-based semantics, and offer high-level operations for them.

Formally, a *split* operation on a value $v_{ik}$ of a parameter $p_i$ to a set of values $\{v_{ik1}, \ldots, v_{ikm}\}$ results in a new domain $L'_{cart}$ defined over $P$ and the set of values $V^*$, where $V^*(p_i) = V(p_i) \setminus \{v_{ik}\} \cup \{v_{ik1}, \ldots, v_{ikm}\}$. This domain is a refinement of the previous domain $L_{cart}$. Similarly, a *merge* operation on a set of values $\{v_{ik1}, \ldots, v_{ikm}\}$ of a parameter $p_i$ to a single value $v_{ik}$ results in a new domain which is an abstraction of the previous one. The connection between the two domains can be described as a Galois connection $(2^{L'_{cart}}, \alpha', \gamma', L_{cart})$ via an extraction function as follows.

**Definition 5 (Galois connection for splitting or merging values).** *Let* $\beta_{ms} : L'_{cart} \to L_{cart}$ *be an extraction function, where* $\forall t \in L'_{cart}, \beta_{ms}(t) = t[v_{ik1}/v_{ik}] \ldots [v_{ikm}/v_{ik}]$. *Then* $\beta_{ms}$ *uniquely defines* $\alpha_{ms}$ *and* $\gamma_{ms}$ *functions as follows:* $\forall T \subseteq L'_{cart}, \alpha_{ms}(T) = \sqcup_{cart}\{\beta_{ms}(t) \mid t \in T\}$, *and* $\forall t \in L_{cart}, \gamma_{ms}(t) = \cup\{t' \in L'_{cart} \mid \beta_{ms}(t') \sqsubseteq_{cart} t\}$.

**Theorem 3.** $(2^{L'_{cart}}, \alpha_{ms}, \gamma_{ms}, L_{cart})$ *is a Galois insertion.*

The proof follows directly from the definitions of $\alpha_{ms}$, $\beta_{ms}$, and $\gamma_{ms}$. For details see [25].

Similarly to the cases described in Sect. 5, the Galois connection is a means to consistently define the new validity function following the domain change, and to expose which new parts of the test space have an unknown validity:

**Definition 6 (Validity function following the split or merge of values).** $\forall t \in L'_{cart}, F'_{valid}(t) = F_{valid}(\alpha_{ms}(\{t\})) = F_{valid}(\beta_{ms}(t))$; $\forall t \in L_{cart}$, $F_{valid}(t) = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in \gamma_{ms}(t)\}$.

Lemma 3 applies to Definition 6 as well. For details see [25].

In case of a split, according to $\beta_{ms}$ definition, we get that if $t(p_i) \notin \{v_{ik1}, \ldots, v_{ikm}\}$ then $F'_{valid}(t) = F_{valid}(t)$. Otherwise, $F'_{valid}(t) = F_{valid}(t')$, where $\forall j \neq i, t'(p_j) = t(p_j)$, and $t'(p_i) = v_{ik}$. That is, the validity of a test with

the split values is inherited from the validity of the same test with the original value.

In case of a merge, we get that $F_{valid}(t) = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in \gamma_{ms}(t)\} = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in L'_{cart} \wedge \beta_{ms}(t') \sqsubseteq_{cart} t\} = \sqcup_{valid}\{F'_{valid}(t') \mid t' \in L'_{cart} \wedge t'[v_{ik1}/v_{ik}] \ldots [v_{ikm}/v_{ik}] \sqsubseteq_{cart} t\}$.

If $t(p_i) \neq v_{ik}$ then we get that $F_{valid}(t) = F'_{valid}(t)$, because in this case $t$ is also in $L'_{cart}$ and the same tests are smaller than $t$ in both domains. Otherwise, $F_{valid}(t) = \sqcup_{valid}\{F'_{valid}(t') \mid \forall j \neq i, t'(p_j) = t(p_j) \wedge t'(p_i) \in \{v_{ik1}, \ldots, v_{ikm}\}\}$. That is, if all split values agree on the validity, the merged value will inherit it. Otherwise, it will be assigned with the $\top_{valid}$ value, forcing the practitioner to explicitly determine it.

In our example from Sect. 2, a split of `OneMonth` into `6To10WorkingDays` and `Over10WorkingDays` will result in the following additions to $F_{valid}$:

- $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Ground}, \texttt{DT} = \texttt{6To10WorkingDays}) = 1$
- $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Ground}, \texttt{DT} = \texttt{Over10WorkingDays}) = 1$
- $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{6To10WorkingDays}) = 0$
- $F_{valid}(\texttt{IS} = \top, \texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{Over10WorkingDays}) = 0$

As previously noted, in our semantics, a test containing a split value (in our example `6To10WorkingDays` or `Over10WorkingDays`), automatically inherits the validity of the same test with the original value (in our example, `OneMonth`), whereas in Boolean semantics, its validity is ambiguous, and depends on the syntactic representation of the constraints.

Now assume the practitioner realized that `OneWeek` and `6To10WorkingDays` are functionally equivalent, and should be merged into a single value named `UpTo10WorkingDays`. As a result of performing the merge as a composite operation in lattice-based semantics, the partial test ($\texttt{IS} = \top, \texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{UpTo10WorkingDays}$) will be assigned with $\top_{valid}$ validity ("unknown"), since the corresponding strongest exclusions and inclusions in the split model "disagree" on its validity. Similarly to the case of adding a value to the model, we suggest to extend the merge operation to include determination of validity for such merged tests, to result in a validity function that satisfies the completeness requirement, i.e., every complete test has a known validity. The practitioner can assign a 0 or 1 validity to the merged test, which is an abstraction of the split strongest exclusions and inclusions in the merged model. Alternatively, the practitioner can decide to keep the unknown validity, and instead specify a 0 or 1 validity for a set of tests $T$ so that the merged test in question is their least upper bound. For example, a 0 validity can be assigned in this case if the item is out of stock, and a 1 validity can be assigned otherwise.

In contrast, in Boolean semantics, after replacing the split values with the merged one, the question whether the combination of values ($\texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{UpTo10WorkingDays}$) appears in the valid set of tests depends entirely on the syntax of the constraints, and there is no indication about the conflicting validity of the split tests with respect to the merged one.

Similarly to split and merge of values, we define corresponding operations for split and merge of parameters. For lack of space, we do not include here the

formal definition of these operations, however it is similar in nature to that of split and merge of values for a single parameter.

## 7 Related Work

There is a large body of work on various aspects of combinatorial testing. To the best of our knowledge, none of it addresses the problem of combinatorial model evolution. Nie et al. [16] survey 93 academic papers on combinatorial testing, and categorize them into 8 different research areas, but do not mention model evolution. There are also many existing CTD tools [4,18,21], which to the best of our knowledge, provide no support for model evolution, though from our practical experience, managing and comprehending changes in models is a challenge encountered frequently by CTD practitioners in their routine tasks.

Qu et al. [20] examine the effectiveness of combinatorial test prioritization and re-generation strategies on regression testing in evolving programs with multiple versions. However, the work ignores the modeling aspects of the evolution.

Much work has been published on the evolution of other kinds of models, from a semantic and syntactic point of view. For example, Maoz et al. presented semantic model differencing for class and activity diagrams [13,14], where the difference between two models is given as a set of diff witnesses, instances of one model that are not instances of the other. We are unaware of any work on model evolution that uses a lattice-based semantics.

Lattices and Galois connections have been widely used in program analysis and verification, based on the abstract interpretation framework [3]. The framework has numerous applications, including for example, in reactive systems [6] and more recently in semantic differencing of programs [19]. Another notable application of lattices is for multi-valued model checking [9,11,15,23]. We are unaware of any use of lattices in the context of model evolution.

## 8 Summary and Future Work

In this work, we demonstrate the shortcomings of the Boolean semantics currently used by CTD tools for reasoning about model evolution, and extend it with a new lattice-based semantics. The new semantics provides a consistent interpretation of changes in the model, and constitutes a new foundation for better comprehension and management of changes in combinatorial models. We further define higher-level atomic operations for combinatorial model evolution using our lattice-based semantics. An analysis of the evolution of 42 real-world industrial models reveals that these operations are indeed recurring evolution patterns, and serves as preliminary evidence for the strength of our semantics.

This work is a first step in a larger research agenda involving different aspects of combinatorial model evolution. We already implemented the new semantics and higher-level constructs in our CTD tool IBM FOCUS. We plan to perform a thorough evaluation of these enhancements and assess the degree to which they help CTD practitioners manage changes in real-world models.

We also plan to extend our analysis of the evolution of real-world models, and identify additional recurring evolution patterns. The current analysis was mostly performed manually, with only lightweight tool support. We plan to further extend the tool support for semantic differencing between combinatorial models.

Another research direction we are pursuing is the co-evolution of models and the test plans derived from them. We plan to use our lattice-based semantics to establish the connection between the interaction coverage requirements in different versions of a model, and determine what changes are required in the test plan to match the evolved model and coverage requirements.

Finally, our work is part of a more general plan to explore the benefits of lattice-based semantics for supporting the evolution of other kinds of models, for example relational models and transition systems.

## References


1. Burroughs, K., Jain, A., Erickson, R.L.: Improved quality of protocol testing through techniques of experimental design. In: SUPERCOMM/ICC (1994)
2. Cohen, M.B., Snyder, J., Rothermel, G.: Testing across configurations: implications for combinatorial testing. SIGSOFT Softw. Eng. Notes **31**(6), 1–9 (2006)
3. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL (1977)
4. Czerwonka, J.: Pairwise Testing in Real World. In: PNSQC (2006)
5. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J.M., Lott, C.M., Patton, G.C., Horowitz, B.M.: Model-based testing in practice. In: ICSE (1999)
6. Dams, D., Gerth, R., Grumberg, O.: Abstract interpretation of reactive systems. ACM Trans. Program. Lang. Syst. **19**(2), 253–291 (1997)
7. Davey, B.A., Priestley, H.A.: Introduction To Lattices and Order. Cambridge University Press, Cambridge (1990)
8. Grindal, M., Lindström, B., Offutt, J., Andler, S.F.: An evaluation of combination strategies for test case selection. Softw. Eng. Empirical **11**(4), 583–611 (2006)
9. Grumberg, O., Lange, M., Leucker, M., Shoham, S.: *Don't know* in the $\mu$-Calculus. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 233–249. Springer, Heidelberg (2005)
10. IBM Functional Coverage Unified Solution (IBM FOCUS). http://researcher.watson.ibm.com/researcher/view_project.php?id=1871
11. Katoen, J.-P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for probabilistic systems. In: JLAP (2012)
12. Kuhn, D.R., Wallace, D.R., Gallo, A.M.: Software fault interactions and implications for software testing. IEEE Trans. Softw. Eng. **30**(6), 418–421 (2004)
13. Maoz, S., Ringert, J.O., Rumpe, B.: ADDiff: semantic differencing for activity diagrams. In: ESEC/FSE (2011)
14. Maoz, S., Ringert, J.O., Rumpe, B.: CDDiff: semantic differencing for class diagrams. In: Mezini, M. (ed.) ECOOP 2011. LNCS, vol. 6813, pp. 230–254. Springer, Heidelberg (2011)
15. Meller, Y., Grumberg, O., Shoham, S.: A framework for compositional verification of multi-valued systems via abstraction-refinement. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 271–288. Springer, Heidelberg (2009)

16. Nie, C., Leung, H.: A survey of combinatorial testing. ACM Comput. Surv. **43**(2), 11 (2011)
17. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, New York (1999)
18. Pairwise testing website. http://www.pairwise.org/tools.asp
19. Partush, N., Yahav, E.: Abstract semantic differencing via speculative correlation. In: OOPSLA (2014)
20. Qu, X., Cohen, M.B., Woolf, K.M.: Combinatorial interaction regression testing: a study of test case generation and prioritization. In: ICSM (2007)
21. Lei, Y., Kuhn, R., Kacker, R.: Practical combinatorial testing beyond pairwise. IT Prof. **10**(3), 19–23 (2008)
22. Segall, I., Tzoref-Brill, R., Farchi, E.: Using binary decision diagrams for combinatorial test design. In: ISSTA (2011)
23. Shoham, S., Grumberg, O.: 3-valued abstraction: more precision at less cost. Inf. Comput. **206**(11), 1313–1333 (2008)
24. Tai, K.C., Lie, Y.: A test generation strategy for pairwise testing. IEEE Trans. Softw. Eng. **1**, 109–111 (2002)
25. Tzoref-Brill, R., Maoz, S.: Lattice-based semantics for combinatorial model evolution. Technical report H-0323, IBM Research (2015)
26. Wojciak, P., Tzoref-Brill, R.: System level combinatorial testing in practice - the concurrent maintenance case study. In: ICST (2014)