Syntactic and Semantic Differencing for Combinatorial Models of Test Designs

Rachel Tzoref-Brill
School of Computer Science, Tel Aviv University
and IBM Research, Israel

Shahar Maoz
School of Computer Science, Tel Aviv University

***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. Manually reasoning about the differences between versions of real-world models following such updates is infeasible due to their complexity and size. Moreover, representing the differences is challenging.**

In this work, we propose a first syntactic and semantic differencing technique for combinatorial models of test designs. We define a concise and canonical representation for differences between two models, and suggest a scalable algorithm for automatically computing and presenting it. We use our differencing technique to analyze the evolution of 42 real-world industrial models, demonstrating its applicability and scalability. Further, a user study with 16 CTD practitioners shows that comprehension of differences between real-world combinatorial model versions is challenging and that our differencing tool significantly improves the performance of less experienced practitioners. The analysis and user study provide evidence for the potential usefulness of our differencing approach.

Our work advances the state-of-the-art in CTD with better capabilities for change comprehension and management.

I. Introduction

One of the effective techniques for coping with the verification challenge of increasingly complex software systems is Combinatorial Test Design (CTD), a.k.a. combinatorial testing [4], [8], [10], [13], [17], [35], [36]. CTD requires a manual definition of the test space in the form of a *combinatorial model*, consisting of a set of parameters, their respective values, and constraints on the value combinations. A valid test in the test space is defined to be an assignment of one value to each parameter that satisfies the constraints. A CTD algorithm automatically constructs a subset of the set of valid tests 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 [10], [18], [31].

An under-explored challenge for wide deployment of CTD in industry is the manual process for modeling and maintaining the test space. In practice, creating a CTD model is not a one time effort; when the system under test evolves, so should the models. In face of the move to agile methodology and to continuous delivery mode, where software development cycles are getting ever shorter, test design needs to frequently adjust to changes, which in the context of CTD means frequent model definition updates. However, although 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. A recent survey by Nie et al. [26] reveals that only around 5% of the publications on CTD explore the crucial modeling process, and the topic of model maintenance is not even mentioned in [26]. Close to 40 CTD tools are listed in [27], e.g., PICT [9], ACTS [19], Jenny [16], and AETG [6], but to the best of our knowledge, none of these 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. Without such tool support, the practitioner is "left in the dark" as to whether the performed change will result in the intended effect, and what other changes may be required. When a series of such change operations is performed, as is typically the case, this problem exacerbates.

In this work, we propose a first approach for syntactic and semantic differencing of combinatorial models. Computing and representing the differences between models are challenging tasks. First, some parts of a model can implicitly change due to explicit changes to other parts, and thus the resulting differences will not be revealed by a purely syntactic comparison. For example, adding a new value to a parameter that appears in the constraints can result in new tests being defined as invalid, even without any explicit changes to the constraints. Second, there can be different syntactic representations to the same model (i.e., representations that result in the same set of valid tests). Specifically, it is challenging to clearly and concisely represent differences between constraints, which are propositional logic formulas over multi-valued parameters. Finally, as our evidence from the field shows, real-world models can be huge, thus computing a semantic differencing must scale well in order to be used in practice.

Our work addresses these challenges by proposing a canonical representation of a combinatorial model, in terms of the value combinations that are excluded from its set of valid tests. The importance of this representation is in its use as a basis for a comparison between models. Other non-canonical representations cannot be used for comparison. Our differencing approach consists of two parts. The syntactic differencing shows the additions and removals of parameters, values, and constraints. The semantic differencing concisely computes and presents the differences in the set of valid tests, based on the canonical representation of the models.

To scale the computation to real-world model sizes, we use an efficient representation of the sets of valid tests and their differences, which is based on Binary Decision Diagrams [3], a compact data structure for representing and manipulating Boolean functions.

It is important to note that differencing of CTD models is *not* a mere theoretical exercise. First, the model, and not the test suite that is directly and automatically derived from it, is the main artifact that CTD test designers create, read, revise, and maintain. Second, the cost of an incorrect model update can be very high: if the model change is incomplete then new tests generated from the model will not adequately cover the software change; if the model change is incorrect it may result in generating redundant or erroneous tests. Finally, as we will demonstrate in Section III, CTD model updates can be quite tricky due to the implicit dependencies between the different parts of the model. Thus, in this work we focus on differencing at the model level. Our differencing technique serves as a review tool for practitioners to verify that their model updates are complete and correct.

We implemented the differencing technique within the industrial-strength commercial CTD tool IBM Functional Coverage Unified Solution (IBM FOCUS) [14], [30]. IBM FOCUS has been in use already for several years by hundreds of CTD practitioners inside and outside of IBM.

To evaluate the applicability of our ideas and implementation to real-world models and their versions, we applied our differencing technique to 42 real-world industrial models with a total of 107 versions (2-5 versions per model, 65 commits). Our analysis reveals that while real-world commits of combinatorial models of test designs tend to be large and complex in practice, our approach provides acceptable performance times in almost all cases.

We further evaluated the effect of our differencing technique on users by conducting a user study with 16 CTD practitioners on two real-world models, each consisting of two real-world versions. The results show that comprehension of differences between real-world combinatorial model versions is challenging and that our differencing tool improves the performance of practitioners. This was most evident for the 8 less experienced practitioners, whose score improved by over 40 percent when using our differencing tool.

To conclude, our contributions are as follows: a first approach for syntactic and semantic differencing of combinatorial models of test designs; a scalable implementation for our differencing technique, implemented in an industrial-strength commercial CTD tool; the application of the new differencing technique to 42 real-world industrial models with a total of 107 versions; and a user study with CTD practitioners that shows that our differencing technique significantly improves the performance of less experienced practitioners in correct comprehension of differences between real-world model versions. The proposed differencing technique advances the state-of-the-art in CTD with new tools for change comprehension and management.

II. Background

We provide background on combinatorial models and their semantics and on the use of binary decision diagrams to represent them.

Combinatorial Models and Their Semantics. A combinatorial model is defined as follows. Let $P = \{p_1, \dots, p_n\}$ be a labelled set of parameters, $V = \{V_1, \dots, V_n\}$ a labelled 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 $(v_1, \dots, v_m)$, where $\forall_i,\ v_i \in V_i$, is a tuple of assignments to the parameters in P.

The semantics used in practice by CTD tools [27] 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 $S(P, V, C)$.

Using Binary Decision Diagrams to Represent Combinatorial Models. In [30], a compact representation of combinatorial models using Binary Decision Diagrams (BDDs) was presented. BDDs [3] are a compact data structure for representing and manipulating Boolean functions, commonly used in formal verification [5] and in logic synthesis [21]. [30] utilizes the efficient computation of Boolean operations on BDDs such as negation, conjunction and disjunction, to compute the BDD representing the set of valid tests from the user-specified constraints. The set of invalid tests in the model is represented using the conjunction of the BDDs for each of the constraints. Multi-valued parameters are handled using standard Boolean encoding and reduction techniques to BDDs [25]. The set of valid tests is represented by the negation of the BDD for the invalid tests, conjunct with a BDD that represents the legal multi-valued to Boolean encodings of the parameter values. This BDD-based representation of the combinatorial model is the basis for the implementation of our semantic differencing.

III. 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 Section IV.

Table I 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.

Below we follow a series of updates to the model, inspired by similar updates we have seen in the evolution of real-world models, adding a value, updating a constraint, splitting a parameter etc. We describe the updates, discuss their semantics,

TABLE I: Example on-line shopping model

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

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

and demonstrate how our differencing solution handles them. The updates are relatively small and local. We use them to demonstrate the basic principles of our analysis, as well as the challenges associated with differencing, even with such seemingly simple updates.

From V1 to V2. Following the addition of a new feature to the system, a practitioner added the value `Sea` to the parameter `OrderShipping`, and committed a second version of the model. One may view this form of change as an *extension*, where parts are added to the model to describe the test space in more detail. Figure 1 depicts the result of our differencing analysis between the first two versions of the model.

The differencing consists of two parts, syntactic and semantic. The syntactic part reports the changes that were made to the parameters, constraints, and values. As expected, in our example it reports solely on the addition of the `Sea` value to the `OS` parameter. The second, semantic part, reports the changes in the set of valid tests, in terms of its *strongest exclusions*. A strongest exclusion is a combination that is excluded by the constraints in the model (and thus does not appear in any valid test), but whose every strict subset is included in the model. The motivation for showing the changes to the strongest exclusions is twofold. First, they constitute a canonical representation of the test space, as we will show in Section IV, and therefore can be used as a basis for comparison between different test spaces. Second, they represent the tests that were excluded from the model in a concise form, and thus can be used to provide information to the practitioner about the differences between the test spaces. Note that while the constraints were not explicitly updated following the addition of the `Sea` value, i.e., syntactically the constraints are identical in the two versions, our analysis reveals that two new strongest exclusions were added to the model: $(\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{Immediate})$, and $(\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{OneMonth})$.

From V2 to V3. When reviewing these added exclusions, the practitioner may have realized that the constraints need to be updated as well, in order to reflect the intended use of the new feature. Thus, she deleted the second constraint and wrote a new one instead: $\texttt{DT} = \texttt{OneMonth} \rightarrow \texttt{OS} = \texttt{Ground} \vee \texttt{OS} = \texttt{Sea}$, and committed a third version of the model. When the differencing analysis is performed between the third and the second versions, the strongest exclusion $(\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{OneMonth})$ will be marked as removed, since it is no longer an exclusion in the third version. One may view this form of change, where a strongest exclusion becomes included in the test space, without being part of a new strongest exclusion, as a *correction*.

From V3 to V4. 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. Thus, she replaced the `OneMonth` value with these two values, deleted the second constraint and wrote a new one instead: $\texttt{DT} = \texttt{6To10WorkingDays} \vee \texttt{DT} = \texttt{Over10WorkingDays} \rightarrow \texttt{OS} = \texttt{Ground} \vee \texttt{OS} = \texttt{Sea}$, and committed a fourth version of the model. Figure 2 depicts the result of our differencing analysis between the third and fourth versions of the model.

One may view this form of change, where a parameter or a value is divided into several different cases, as a *split*. The split of the value is displayed on the left side, and the split of the strongest exclusion is displayed on the right side, where the original one is removed and replaced with new strongest exclusions, one for each case. The split pattern is easy to detect when viewing the changes to the strongest exclusions, whereas it becomes less obvious when viewing the removed and added constraints. Furthermore, a constraint can have different syntactic representations that are semantically equivalent. For example, the second constraint in the fourth version could have been $\texttt{DT} \neq \texttt{Immediate} \wedge \texttt{DT} \neq \texttt{OneWeek} \rightarrow \texttt{OS} = \texttt{Ground} \vee \texttt{OS} = \texttt{Sea}$, resulting in a syntactically different yet semantically equivalent model. The model with this representation of the second constraint would completely hide the fact that it splits `OneMonth` into two new values, as these values do not even appear in the constraint. Moreover, this version of the constraint can be used also in the third version of the model, in which case no explicit changes would have been made to the constraints when moving to the fourth version, though the split of the strongest exclusions would still occur.

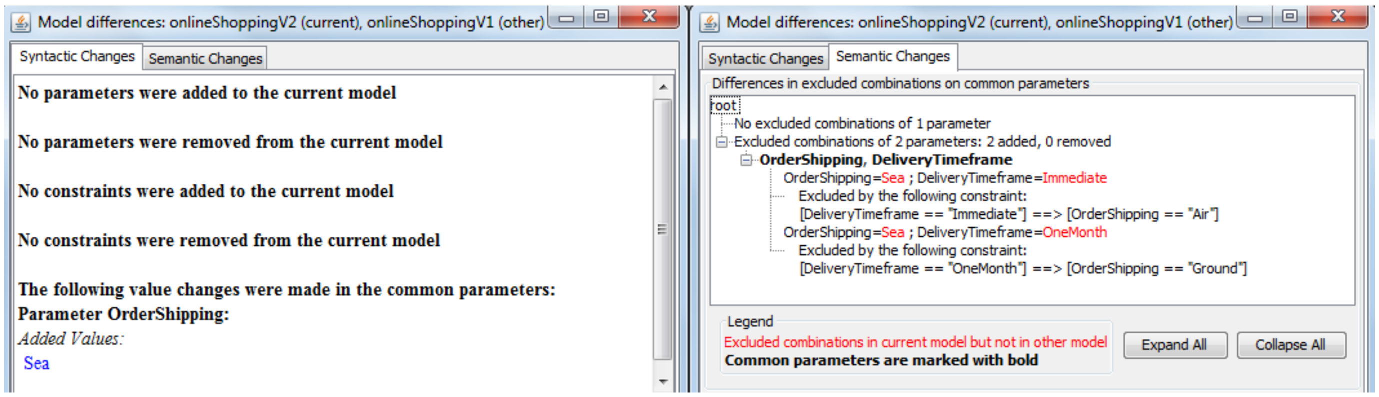
IV. Formal Solution for Model Differencing

We now formally present our proposed differencing technique for combinatorial models, followed by a description of our algorithm for computing it. Throughout this section we will use the notations defined in Section II and demonstrate the ideas on the running example presented in Section III.

A. Model Differencing Definitions

Given a combinatorial model $S(P, V, C)$, for every parameter $p \in P$, we define $p.name$ to be its identifier. Similarly, for every $V_i \in V$, $v.name$ denotes the identifier of every value $v \in V_i$. We define corresponding sets for the identifiers of the parameters and their values, $PN = \{p.name \mid p \in P\}$ and $\forall V_i \in V, VN_i = \{v.name \mid v \in V_i\}$. For every constraint $c \in C$, we define $c.expr$ to be its Boolean expression. We define the set $CE = \{c.expr \mid c \in C\}$ for the expressions of the constraints.

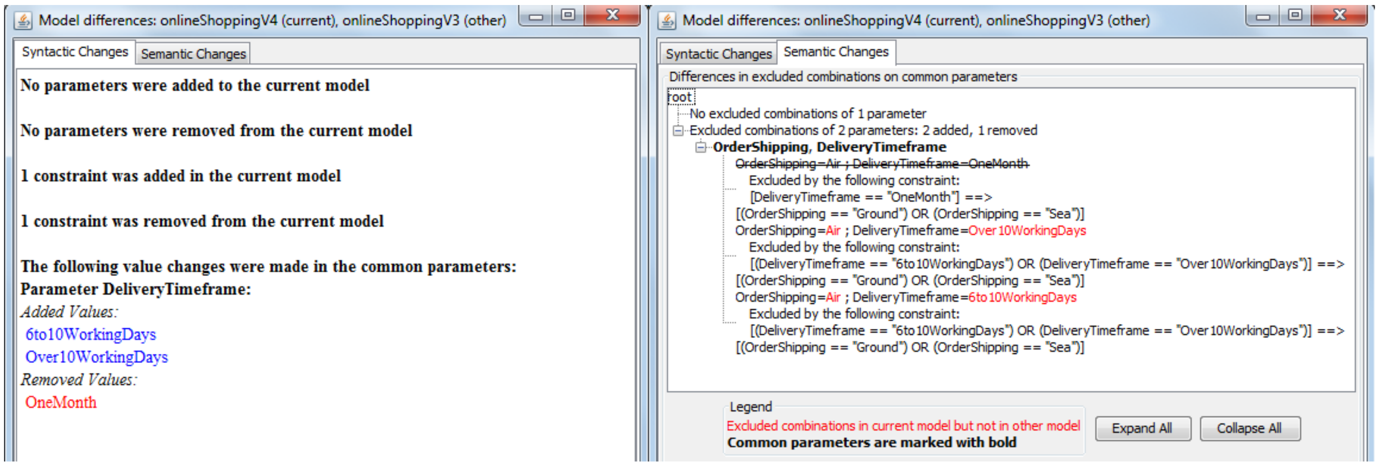
Given two models $S^1(P^1, V^1, C^1)$ and $S^2(P^2, V^2, C^2)$, we define a syntactic differencing $Diff_{syn}(S^1, S^2)$ and a



```
Model differences: onlineShoppingV2 (current), onlineShoppingV1 (other)
Syntactic Changes | Semantic Changes

No parameters were added to the current model

No parameters were removed from the current model

No constraints were added to the current model

No constraints were removed from the current model

The following value changes were made in the common parameters:
Parameter OrderShipping:
Added Values:
 Sea
```

```
Model differences: onlineShoppingV2 (current), onlineShoppingV1 (other)
Syntactic Changes | Semantic Changes
Differences in excluded combinations on common parameters
root
  No excluded combinations of 1 parameter
  Excluded combinations of 2 parameters: 2 added, 0 removed
    OrderShipping, DeliveryTimeframe
      OrderShipping=Sea ; DeliveryTimeframe=Immediate
        Excluded by the following constraint:
        [DeliveryTimeframe == "Immediate"] ==> [OrderShipping == "Air"]
      OrderShipping=Sea ; DeliveryTimeframe=OneMonth
        Excluded by the following constraint:
        [DeliveryTimeframe == "OneMonth"] ==> [OrderShipping == "Ground"]

Legend
Excluded combinations in current model but not in other model
Common parameters are marked with bold
```

Fig. 1: Differencing between the first and second versions. On the left, the syntactic differencing of parameters, constraints, and values. On the right, the semantic differencing of strongest exclusions.



```
Model differences: onlineShoppingV4 (current), onlineShoppingV3 (other)
Syntactic Changes | Semantic Changes

No parameters were added to the current model

No parameters were removed from the current model

1 constraint was added in the current model

1 constraint was removed from the current model

The following value changes were made in the common parameters:
Parameter DeliveryTimeframe:
Added Values:
 6to10WorkingDays
 Over10WorkingDays
Removed Values:
 OneMonth
```

```
Model differences: onlineShoppingV4 (current), onlineShoppingV3 (other)
Syntactic Changes | Semantic Changes
Differences in excluded combinations on common parameters
root
  No excluded combinations of 1 parameter
  Excluded combinations of 2 parameters: 2 added, 1 removed
    OrderShipping, DeliveryTimeframe
      ~~OrderShipping=Air ; DeliveryTimeframe=OneMonth~~
        Excluded by the following constraint:
        [DeliveryTimeframe == "OneMonth"] ==>
      [(OrderShipping == "Ground") OR (OrderShipping == "Sea")]
      OrderShipping=Air ; DeliveryTimeframe=Over10WorkingDays
        Excluded by the following constraint:
        [(DeliveryTimeframe == "6to10WorkingDays") OR (DeliveryTimeframe == "Over10WorkingDays")] ==>
      [(OrderShipping == "Ground") OR (OrderShipping == "Sea")]
      OrderShipping=Air ; DeliveryTimeframe=6to10WorkingDays
        Excluded by the following constraint:
        [(DeliveryTimeframe == "6to10WorkingDays") OR (DeliveryTimeframe == "Over10WorkingDays")] ==>
      [(OrderShipping == "Ground") OR (OrderShipping == "Sea")]

Legend
Excluded combinations in current model but not in other model
Common parameters are marked with bold
```

Fig. 2: Differencing between the third and fourth versions. On the left, the syntactic differencing of parameters, constraints, and values. On the right, the semantic differencing of strongest exclusions.

complete semantic differencing $\mathit{Diff}_{csem}(S^1, S^2)$ between the two models as follows.

1) Syntactic Differencing: $\mathit{Diff}_{syn}(S^1, S^2)$ consists of the following parts:

1) Parameter additions: $\{p \mid p.name \in PN^1 \setminus PN^2\}$
2) Parameter removals: $\{p \mid p.name \in PN^2 \setminus PN^1\}$
3) Constraint additions: $\{c \mid c.expr \in CE^1 \setminus CE^2\}$
4) Constraint removals: $\{c \mid c.expr \in CE^2 \setminus CE^1\}$
5) Value additions: $\bigcup_{p_i \in P^1, p_j \in P^2}\{v \mid v \in V^1{}_i \wedge v.name \in VN^1_i \setminus VN^2_j \wedge p_i.name = p_j.name\}$
6) Value removals: $\bigcup_{p_i \in P^1, p_j \in P^2}\{v \mid v \in V^2{}_j \wedge v.name \in VN^2_j \setminus VN^1_i \wedge p_i.name = p_j.name\}$

For example, the left side of Figure 2 presents the syntactic differencing between $V3$ and $V4$ of the on-line shopping model, which consists of the addition of two values to the common parameter `DT`, the removal of one value from it, no parameter additions or removals, the addition of one constraint, and the removal of another.

2) Semantic Differencing: While syntactic differencing reasons about the syntactic changes in the parameters and values, complete semantic differencing reasons about all the differences in the test space of S^1 and S^2.

To enable a comparison of the two test spaces, we first define a concise and canonical representation of a test space. The representation we propose is based on the notion of *strongest exclusions*. A strongest exclusion is a combination that is excluded by the constraints in the model (and thus does not appear in any valid test), but whose every strict subset is included in the model and thus appears in at least one valid test. The set of strongest exclusions of a model, together with the set of parameters and their values, uniquely define the test space of the model.

Theorem 4.1: Let SE be the set of strongest exclusions of a combinatorial model $S(P, V, C)$. Then (P, V, SE) is a canonical representation of $S(P, V, C)$.

Proof: We will show that for two models $S^1(P, V, C^1)$ and $S^2(P, V, C^2)$, if $S^1 = S^2$ (they have the same set of valid tests), then $SE^1 = SE^2$. Assume to the contrary, i.e., that there exists a strongest exclusion $e \in SE^1$ such that $e \notin SE^2$. Then it follows from the definition of a strongest exclusion that one of the following holds: (1) e strictly contains a strongest exclusion $e' \in SE^2$. Since $S^1 = S^2$, it must hold that e'

is excluded from S^1, contrary to our initial assumption that $e \in SE^1$. (2) e is not an exclusion in S^2, which means there is a test $t \in S^2$ that contains e. Since $S^1 = S^2$, it holds that $t \in S^1$, contrary to our initial assumption that $e \in SE^1$. ■

For example, for the on-line shopping model from Section III, $SE^{V1} = \{(\texttt{OS} = \texttt{Air}, \texttt{DT} = \texttt{OneMonth}), (\texttt{OS} = \texttt{Ground}, \texttt{DT} = \texttt{Immediate})\}$. Note that each strongest exclusion represents multiple complete tests that are excluded from the model. Note also that each constraint defines one or more strongest exclusions.

$\mathit{Diff}_{csem}(S^1, S^2)$ consists of the following parts:

1) Strongest exclusion additions: $\{e | e \in SE^1 \setminus SE^2\}$;
2) Strongest exclusion removals: $\{e | e \in SE^2 \setminus SE^1\}$.

For example, for the on-line shopping model, $SE^{V2} = SE^{V1} \cup \{(\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{Immediate}), (\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{OneMonth})\}$. Hence, $\mathit{Diff}_{csem}(V2, V1)$ contains the latter two strongest exclusions, as depicted on the right side of Figure 1.

The canonical representation of a model using strongest exclusions allows us to define the *completion level* (CL) of a version of a model, as well as of a commit (differencing two model versions). The CL of a model version is the maximal arity of a single strongest exclusion. It indicates the maximal depth of exclusions that needs to be explored in order to reach a complete canonical representation of the model.

In our running example, the CL of $V5$ is 3; $V5$'s canonical representation includes no strongest exclusions of arity larger than 3.

CL is extended naturally from a single model version to the comparison of a commit consisting of two model versions. Specifically, the CL of a commit is the maximum between the CLs of the two model versions, before and after the commit. It represents an upper bound on the maximal number of parameter values appearing in all strongest exclusions required for presenting all the commit differences. The CL of a commit is only an upper bound on the largest strongest exclusion that will appear in the diff, because the diff contains only strongest exclusions that are not common to both model versions. As in the single model case, we use the CL of a commit as an indicator for the maximal depth that needs to be explored in order to reach a complete differencing between two model versions. In Section IV-B, we present a way to compute the CL of a model (and of a commit).

While presenting the differences in terms of strongest exclusions is purely semantic and independent of the syntactic representation of the model, it is also beneficial to link semantic information to syntactic information. As shown in Figures 1 and 2, we achieve this information linking by mapping the strongest exclusions that appear in the differencing back to their excluding constraints, as specified by the CTD practitioner. Each strongest exclusion in $\mathit{Diff}_{csem}(S^1, S^2)$ serves as a witness for the test space change induced by its excluding constraint under the changes to the model parameters and values. For example, in Figure 1, the strongest exclusion $(\texttt{OS} = \texttt{Sea}, \texttt{DT} = \texttt{Immediate})$ is mapped to its excluding constraint $\texttt{DT} = \texttt{Immediate} \rightarrow \texttt{OS} = \texttt{Air}$. The information that this constraint is responsible for the added strongest exclusion is particularly valuable in this case, since no explicit changes were made to the constraints in version $V2$ to exclude this combination, following the addition of the `Sea` value.

Discussion of Alternatives. A rather naïve approach for (partial) semantic differencing could have been to present complete tests that are valid in one version of the model and not in the other. One problem with this approach is that since the versions may have different parameters and values, the question whether a test in one version is valid in another version may not be well defined, i.e., the constraints in one version may refer to parameters and values that do not exist in the other version, and vice versa. Another problem with this naïve approach is that it must be partial, since the number of such complete tests may be huge. Our solution avoids these two problems. First, by computing the strongest exclusions in each model, and then comparing the two resulting sets of value combinations, our differencing is well defined. Second, since each strongest exclusion represents numerous complete tests that are excluded from the model, our solution concisely represents all the test space differences between the two models, and is orders of magnitude smaller than one that relies on complete tests.

Another alternative approach to the use of strongest exclusions for semantic differencing could be to use partial inclusions or strongest inclusions. Partial inclusions are value combinations that are contained in at least one valid test. Strongest inclusions are value combinations whose every extension to a complete test forms a valid test. However, both options of using inclusions are significantly less informative than using strongest exclusions. Typically, all parameter values of a model are valid in combination with at least one assignment to the other parameters. Hence, all model values are partial inclusions. A diff based on partial inclusions would simply result in the list of added and removed values, making the semantic differencing identical to our syntactic one. Moreover, partial inclusions do not form a canonical representation of a model. Strongest inclusions form a canonical representation, but are usually complete tests. Thus, a diff based on strongest inclusions will result in a significantly larger and less concise $\mathit{Diff}_{csem}(S^1, S^2)$ than when using strongest exclusions.

3) Reducing the size of the semantic differencing: While a complete semantic differencing is desirable, it is also beneficial to reduce the amount of strongest exclusions that the practitioner needs to review. Two techniques to achieve this are filtering out redundant information, and dividing the presented information into categories.

Derived Exclusions. Redundant information may exist in $\mathit{Diff}_{csem}(S^1, S^2)$ due to *derived exclusions*. A derived exclusion is an invalid value combination (i.e., it is excluded from the model) that is excluded not due to any single constraint but rather due to the interaction between different constraints. For example, consider the first constraint in Table I, and assume we add a third constraint to the model: $\texttt{OS} = \texttt{Air} \rightarrow \texttt{IS} = \texttt{InStock}$. The interaction of the two constraints yields a derived exclusion, $(\texttt{DT} = \texttt{Immediate}, \texttt{IS} = \texttt{OutOfStock})$, which is not directly excluded by any of the three constraints.

Some of the strongest exclusions that appear in the diff may be derived exclusions, and thus contain redundant information – the fact that they are excluded can be concluded from other excluded combinations in the diff. To eliminate this redundancy and reduce the amount of information in $\mathit{Diff}_{csem}(S^1, S^2)$, we remove derived exclusions from the result of the comparison between SE^1 and SE^2.

Note that due the removal of derived exclusions, Diff_{csem} (S^1, S^2) is no longer canonical, since the same exclusion can be derived in one model and not in another semantically-equivalent model (i.e., that has the same set of valid tests), depending on the syntactic representation of the constraints. We note however that the importance of strongest exclusions as a canonical representation of a model is for comparison purposes (other non-canonical representations cannot be used for comparison), and reduction to a non-canonical form is done only on the diff results after the comparison, for presentation purposes.

Removing derived exclusions has two important advantages. First, it enables tighter linkage of the semantic information to syntactic information, because each strongest exclusion presented in $\mathit{Diff}_{csem}(S^1, S^2)$ can be linked to a single user-specified constraint that excludes it, as presented in Figures 1 and 2. Second, it filters out redundant information from the presentation and reduces the number of exclusions that the practitioner needs to review.

Categorization. Another technique that helps with the analysis of the diff is to divide the computed information into categories. We observe that most of the analysis effort concentrates on the strongest exclusions that contain common parameters, because they indicate changes in the parts of the test space that belong to both model versions. When a set of parameters is removed from the model, it is natural that all the constraints on their inter-relations are also removed from the model, and similarly for the case of parameter additions.

To this end, and for clarity of presentation, we divide the semantic diff view into three separate views: (1) additions and/or removals that are defined on at least one common parameter, which is the main diff view, (2) additions that are defined only on added parameters, and (3) removals that are defined only on removed parameters.

In all our examples from Section III, all exclusions differences are of the first type, hence only the main view is shown. Note that an added or removed strongest exclusion may be defined both on common parameters and on parameters unique to one model, in which case it will be only presented in the main view.

B. Computing the Differencing

While computing $\mathit{Diff}_{syn}(S^1, S^2)$ is straightforward and can be achieved by a simple traversal over the parameters, values, and constraints of the two versions, computing $\mathit{Diff}_{csem}(S^1, S^2)$ is much more challenging. Specifically, it requires computing the set of strongest exclusions SE of a model version. To achieve efficient computation of SE, we use a symbolic computation based on Binary Decision Diagrams (BDDs) [3] as our primary data structure for the set of valid tests as well as for all other computed artifacts. As explained in Section II, to handle domains of discrete values rather than only Boolean ones, we use a BDD that represents the legal multi-valued to Boolean encoding of the parameter values. For clarity of presentation, we omit the handling of this standard encoding from the following pseudo code and accompanying algorithm description.

```
input : The BDD $Valid_{S^1}$ of all valid tests of $S^1$.
        The BDD $Valid_{S^2}$ of all valid tests of $S^2$.
        The set of constraints $C^1$ of $S^1$.
        The set of constraints $C^2$ of $S^2$.
output: The BDD $seAdded$ of strongest exclusions that are in $S^1$
        and not in $S^2$. The BDD $seRemoved$ of strongest exclusions
        that are in $S^2$ and not in $S^1$.
1  Init: $completed1 \leftarrow FALSE$
2  $completed2 \leftarrow FALSE$
3  $SE^1 \leftarrow newlist$
4  $SE^2 \leftarrow newlist$
5  $level \leftarrow 1$
6  while $\neg(completed1 \wedge completed2)$ do
7      for $i \in 1, 2$ do
8          if $\neg completedi$ then
9              computeSE($Valid_{S^i}, level, SE^i$)
10             if $\bigvee(SE^i) == \neg Valid_{S^i}$ then
11                 $completedi \leftarrow TRUE$
12             end
13         end
14     end
15     $level++$
16 end
17 $seAdded \leftarrow SE^1 \setminus SE^2$
18 $seRemoved \leftarrow SE^2 \setminus SE^1$
19 removeDerived(seAdded, $C^1$)
20 removeDerived(seRemoved, $C^2$)
```
Algorithm 1: Semantic diff of combinatorial models

In Algorithm 1, we introduce a novel algorithm for computing the semantic differencing of two model versions. The algorithm is iterative, where in each step it computes the strongest exclusions up to arity $level$ (line 9), until reaching the CL. Once reached, it compares the two sets of strongest exclusions to identify their differences (l. 17), and removes derived exclusions from these differences (l. 19). BDDs are used to represent the set of valid tests of each model version ($Valid_{S^i}$), to compute and represent the strongest exclusions of each version (SE^1 and SE^2) and their differences ($seAdded$ and $seRemoved$), to check whether the CL has been reached (l. 10), and to remove the derived exclusions from the differencing results. The algorithm relies on the efficiency of negation, conjunction, disjunction, and existential quantification of BDDs, as explained in Section II.

Algorithm 2 presents the method computeSE, which computes in each iteration the strongest exclusions of arity $level$ for a given model. For every set of parameters t of size $level$, it computes the BDD $excluded_t$ of all exclusions on t. This is achieved by first computing the BDD of valid assignments to t, which is the projection of the $Valid$ BDD on t, and then negating the resulting BDD (l. 3). A projection on a set of parameters t is computed by existentially quantifying out the parameters not appearing in t.

Next, to filter out exclusions on t that are not strongest exclusions, the method conjuncts the BDD $excluded_t$ with the negation of every strongest exclusion BDD from a level smaller than $level$ whose parameters are contained in t (l. 7). Thus, all strongest exclusions for each set of parameters t are symbolically computed at once using BDD operations.

At the end of each call to `computeSE`, the main algorithm checks whether the CL has been reached, by checking whether the strongest exclusions collected so far represent the entire valid test space. This is achieved by checking whether the disjunction of the strongest exclusions collected so far is identical to the BDD of invalid tests (l. 10).

Finally, once the CL of both models is reached (and hence the CL of the commit), the iteration in Algorithm 1 terminates, and the set of differences is computed. The derived exclusions are removed from the result in the `removeDerived` method. This is also computed symbolically for every set of parameters t of size up to CL, by projecting each of the constraints on t, and removing from the BDD of strongest exclusions for t all tuples that are not included in one of these projections, using BDD conjunction and disjunction operations. For additional details about the computation of derived exclusions see [11]. Note that the removal of the derived exclusions cannot precede the identification of the exclusion differences, because the same strongest exclusion can be derived in one version and not the other. Had the two operations been swapped, such an exclusion would be falsely identified as a difference between the two versions.

In practice, for test spaces with a huge number of parameters and high CLs, `computeSE` as described so far might require large memory or long computation time for computing the projections of the legal space BDD on all parameter tuples of size $level$. This step (Line 3 of `computeSE`) is the main contributor to the complexity of our algorithm, since for each model it requires $\binom{n}{k}$ project operations on the BDD of the valid test space, where n is the number of parameters, and k is the CL. Thus, we choose to limit the size of the BDDs used during this computation. If the intermediate BDDs exceeds a given size threshold, the iteration on the parameter tuples is interrupted, and the result achieved so far from the previous level is used instead. In such cases, the user will be notified that the differencing is incomplete, and that differences are shown only for strongest exclusions up to the last fully computed level. In Section V, we will show that most real-world models we analyzed reach the CL. Of course, a higher threshold could be used to allow the remaining models reach their CL.

```
input : The BDD $Valid_S$ of all valid tests of $S$.
        The arity $level$ of strongest exclusions to compute.
        The list of BDDs $SE$ of strongest exclusions for
        levels smaller than $level$.
output: The list of BDDs $SE$ of strongest exclusions for levels up to and
        including $level$.
1  Init: $T \leftarrow$ all parameter tuples of size $level$
2  for $t \in T$ do
3      $excluded_t \leftarrow \neg$project($Valid_S, t$)
4      $sizeSE \leftarrow$ size($SE$)
5      for $i \in 0, \dots, sizeSE - 1$ do
6          if $t \supseteq param(SE(i))$ then
7              $excluded_t \leftarrow excluded_t \wedge \neg SE(i))$
8          end
9      end
10     $SE \leftarrow \{SE, excluded_t\}$
11 end
```

Algorithm 2: `computeSE`

V. Evaluation

We present an evaluation of our work in terms of the results of our differencing technique when applied to real-world model evolution. We then continue with a user study evaluating the effectiveness of our differencing technique in helping CTD practitioners comprehend model changes.

A. Real-World Evidence

The research questions guiding our first evaluation are:

RQ1A How do combinatorial models evolve in practice? In particular, what are the common kinds and sizes of changes?

RQ1B How does our differencing computation perform on real model versions in practice?

To answer these questions, we applied our differencing tool to the evolution of a large corpus of real-world models.

1) Models Used and Setup: We applied our differencing technique to the evolution of 42 real-world industrial models, with 107 versions and 65 version commits[1]. The models were written by different CTD practitioners over a period of 7 years, and originate from 12 different domains: firmware, PaaS, IaaS, file system, operating system, database, storage, analytics, banking, telecom, networking, and software applications (email, document management, finance, etc.). The models also capture different levels of testing, such as function test, system test, etc. We did not select the models according to any criteria, and they represent all data available to us. The versions result from user-defined commits; the time difference between two consecutive versions ranges between a day and 13 months. All runs of our differencing computation were performed on a Linux machine with 16 1.5 GHz cores and 16 GB RAM. Only one core was used in our experiments. The BDD package used was JDD [15].

In the following we provide observations on the collected model data and on our differencing results. Complete per model data as well as all versions data and differencing results are available from [2].

2) Results and Observations: **Variability in Size and Complexity.** The data shows high variability in size and complexity of both versions and commits. The size of models ranges from 4 to 109 parameters with median 11.75 and standard deviation (SD) of 13.78 (median and SD are computed on average across commits per model). There were 2 to 500

[1]Unfortunately, all models are confidential since they were created for IBM or its clients. We are in the process of checking the option of sharing most of them after obfuscation.

values per parameter (median: 4.4, SD: 2.73), and 0 to 381 constraints (median: 13.6, SD: 46.7).

The resulting test space sizes are ranging from 69 to $3*10^{42}$ valid tests (median: $10^{6.05}$, SD: $10^{7.6}$), and CLs ranging from 0 to 9 (median: 2.5, SD: 1.55). The size of the changes resulting from commits ranges from the addition or removal of a single parameter, constraint and/or value to 56 parameters added (median: 2, SD: 7.2) and 58 removed (median: 1, SD: 6.4), 266 constraints added (median: 9, SD: 34.8) and 375 removed (median: 5.15, SD: 29.2), 112 values added to a common parameter (median: 1.85, SD: 17.9) and 205 removed (median: 1.55, SD: 31.27), and from no changes in the strongest exclusions to 3096 strongest exclusions added (median: 19.25, SD: 228.5) and 266 removed (median: 8.75, SD: 29.05).

While 50% of the models had rather small (2 or less) parameter additions on average across commits, 25% of the models had 7.6 or more parameter additions on average, and while 50% of the models had 1 or less parameter removals on average, 25% had 4.3 or more such removals on average. Similarly, while 50% of the models had less than 2 value additions to common parameters on average, 25% had 9.5 or more such additions on average, and while 50% of the models had less than 2 value removals on average, 25% had 10 or more such removals on average.

Syntactic Diff. Based on the observed data, changes in constraints (additions in 82% of the commits, removals in 71% of the commits) are more likely to occur than the other syntactic changes, which are more or less equally common (parameter/value additions as well as value removals in 65% of the commits, parameter removals in 55% of the commits).

Additions and Removals. When considering both syntactic and semantic diff, 89% of the commits included both additions to the model and removals from it. 8% included only removals, and 3% only additions. We conclude that a commit tends to contain complex changes rather than simpler changes of only additions to the model or only removals from it.

To answer [RQ1A], the results show high variability in size and complexity of both versions and commits. Commits tend to contain large and complex changes that typically involve a combination of additions, removals, and modifications to the constraints.

The large size and complexity of the models and changes involved, as evident above, require an efficient and scalable differencing solution to handle real-world model evolution.

Performance. Figure 3 presents the runtime results of our semantic differencing computation. Out of a total of 65 commits, computing differencing (in the tool, to display the differences as shown in the screenshots from our running example) took less than a second for 45 of the models (70%), less than 10 seconds for 53 (82%), and only 4 required over a minute (6% of the commits, from only two models). These two latter models involved an exceptionally high number of parameters, large test space BDDs, and a CL greater than 2, all contributing to the exceptionally long running time.

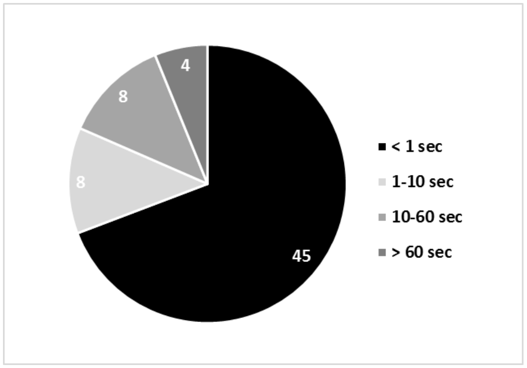


Fig. 3: Performance of our semantic differencing on real-world model commits

To answer [RQ1B], the results show that our differencing computation performs in acceptable times on almost all real model versions in practice.

B. User Study

The research questions guiding our user study are:

RQ2A Is comprehension of changes in models challenging, and does our presentation of syntactic and semantic differencing help practitioners in better understanding the changes done and their consequences?

RQ2B Does the expertise level influence the performance and confidence of practitioners when attempting to understand such changes?

RQ2C Is there a difference in difficulty of comprehension between syntactic and semantic changes?

1) Setup and Participants: Our study included two real-world models, each with two real-world versions. The first model, model A, describes a system test space for features of IBM® POWER7® [36]. Its first version contains 7 parameters and 33 constraints, and its second version contains 6 parameters and 24 constraints. The second model, model B, describes an end-to-end test space for a PaaS. Its first version contains 19 parameters and 2 constraints, and its second version contains 21 parameters and 4 constraints. The sizes and changes in the two models are comparable to real-world model sizes and changes, as reflected by the evidence shown in Section V-A.

We asked 5 questions about each model, related to the changes between the two versions. One question about the syntactic differences between the two versions, and 4 questions about the semantic differences. The syntactic questions asked about the number of parameters or values that were added or removed from the second version.

We presented two types of semantic questions. The first type referred to the effect of adding values or parameters to the model, e.g., *"By adding the value v to parameter X in the model, how many new combinations of v and a value u of parameter Y are added to the space of valid tests?"*.

The second type of semantic question referred to specific value combinations and whether there was a change in their validity status, e.g., *"The value combination C is excluded from the valid test space of version 1; is it also excluded from version 2?"*.

Before running the study, we reviewed these questions with the two CTD practitioners who created the two models we used in our study, and verified that they reflect real-world comprehension tasks of model changes.

Each study participant filled an online questionnaire in an IBM corporate network, containing the questions about the two models (10 questions in total). For one model only the model version definitions were provided, and for the other model the syntactic and semantic diff report was provided as well (as shown in Section III). The decision on the order of models in the questionnaire as well as on the identity of the model that included the differencing report was randomly made per participant. In addition, after each question the participants were asked to rate their confidence in the correctness of their answer from 1 (least) to 5 (most). 16 practitioners participated in the study, all of whom are CTD practitioners regularly involved in creating and comprehending combinatorial models for test designs, as part of their work at IBM. All participants were previously unfamiliar with the two study models.

2) Results: We separate the question scores and confidence into two groups: questions answered without using our differencing report (score/confidence without diff), and questions answered while using our differencing report (score/confidence with diff). Note that due to the setup explained above, some participants had the differencing report available for model A, while others had it for model B. Similarly, some participants had the differencing report available for the first model they were asked about (be it A or B) while others had it for the second one.

Figure 4 summarizes the score results of our study. The average score without diff was 68.75 (out of 100), and with diff 76.25. The average confidence without diff was 4.11 (out of 5) and with diff 4.16. These numbers indicate an overall improvement of 10% in performance and a slight improvement in confidence when using the differencing report.

To answer [RQ2A], the results show that comprehension of changes in models is challenging, and that our presentation of differencing improves practitioners comprehension of the changes done and their consequences.

A deeper analysis is required in order to understand the impact of the differencing report on different practitioners. To this end, we explore the differences between the practitioners based on their level of expertise, as follows. The overall average score was 72.5. The 16 participants are composed of two distinct groups. 8 participants are expert CTD practitioners with a high level of expertise. All these participants scored between 80 and 100, much above the overall average score. 8 participants are less experienced ones with a relatively low level of expertise in CTD. These participants scored between 40 and 70, all below the overall average score.

Figure 4 summarizes also the score results of the expert practitioners and less experienced practitioners, separately. For the expert practitioners, the average score was 95 without the diff report and 92.5 with it. Out of 40 questions per model collectively, the experts answered correctly 37 with diff and 38. We consider this difference negligible. Thus, the expert practitioners performed very well regardless of the diff. Their average confidence was 4.42 without the diff report and 4.55 with it, indicating only a very slight increase in confidence. In contrast, for the less experienced practitioners, the average score was 42.5 without the diff report and 60 with it, indicating a significant improvement in performance. Their average confidence was 3.8 without the diff report and 3.78 with it, relatively low in both cases, and rightfully so.

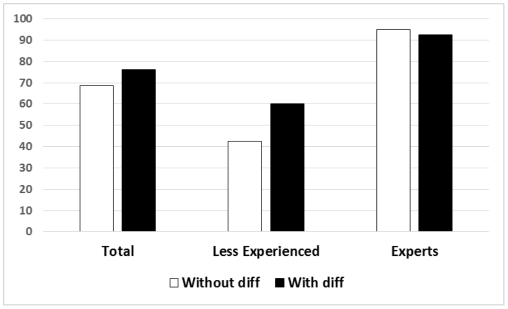


Fig. 4: Average score for the total 16 practitioners, as well as separately for the 8 experts and 8 less experienced practitioners.

To answer [RQ2B], for less experienced practitioners, the differencing report significantly improves performance while not impacting confidence; for expert practitioners, the diff report does not impact performance and confidence.

All participants answered all the syntactic questions correctly, regardless of their level of experience and regardless of whether the diff report was available. For the semantic questions alone, the average score was 60.94 without the diff report and 70.31 with it. The average confidence for a syntactic question was 4.34, above the overall average confidence of 4.14.

To answer [RQ2C], the results show, as one might expect, that syntactic questions are much easier to answer than semantic ones, hence our differencing technique is more useful for understanding semantic differences.

The complete study results are available from [2].

C. Threats to Validity

We discuss threats to the validity of our results, starting with internal validity. First, our implementation of differencing computation may not be free of bugs. To mitigate this, we manually verified the results of the diff computation on small synthetic models and on many of the real-world models. Second, there could have been a bias in the results of our user study due to the order of models as well as the order between questions for which the diff report was provided and those for which it was not provided. To mitigate this, as explained above, we randomized the order of models in the user study, as well as the decision whether the first or second model would be accompanied with the diff report.

There are also threats to external validity. First, the models and versions used in the user study and in the analysis might not be representative of real-world model evolution. To eliminate this threat, we chose real-world models and versions, all that were available to us. Second, our study participants might not be representative of real-world practitioners. To mitigate this, we chose 16 real-world CTD practitioners who are regularly involved in creating and comprehending combinatorial models for test designs. However, a larger group of practitioners and more models and questions could strengthen the generalizability of our results.

VI. Related Work

Much work has been published on syntactic and semantic differencing of programs and models within the more general field of Software Evolution (see, e.g. [12], [28], [37]).

Most relevant to our present work is the work of Maoz et al. on semantic model differencing for Class Diagrams [24] (using SAT) and Activity Diagrams [23] (using BDD-based symbolic algorithm). Other recent work considered semantic differencing for Feature Models [1], [32], in the context of software product lines. These works address similar challenges to the ones we deal with, including the efficient computation of the semantic differences and their effective presentation to the engineer. A framework for relating syntactic and semantic model differences has been presented in [22]. To the best of our knowledge, our work is the first to consider syntactic and semantic differencing for combinatorial models. In addition, the existing model differencing work provided prototype implementations and evaluated performance, but none has evaluated the usefulness of diff representations to real-world practitioners.

While 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 differencing of combinatorial models in the context of their evolution. The survey by Nie et al. [26] considers 93 academic papers on combinatorial testing but does not mention model evolution. Many CTD tools [6], [9], [16], [19], [27] exist, but to the best of our knowledge, they provide no support for model differencing. Our practical experience, in contrast, shows that managing and comprehending changes in models is a challenge encountered frequently by practitioners. One notable exception is the work of Qu et al. [29], which examines the effectiveness of combinatorial testing prioritization and re-generation strategies on regression testing in evolving programs with multiple versions. However, the work does not address evolution at the model level.

Derived exclusions in combinatorial models, a.k.a. *implicit constraints*, have been discussed in previous works and were shown to complicate the solving of the CTD problem [20] and the modeling process [7]. [11] uses derived exclusions to review and debug the model constraints. While it suggests to present only minimal derived exclusions, the concept of strongest exclusions is not discussed independently. Moreover, the work does not suggest the use of strongest exclusions for a canonical representation of a model. Finally, all these works do not consider derived exclusions in the context of evolution.

In a recent work, we presented a lattice-based semantics for interpreting the evolution of combinatorial models [33]. The new semantics replaces the inadequate Boolean semantics which is currently in use by CTD tools for interpreting model changes. It provides a theoretical base for a consistent interpretation of atomic changes to the model, and exposes which additional parts of the model must change following an atomic change, in order to restore validity. The differencing approach we present and evaluate in this paper fits well with this new semantics, since strongest exclusions can be precisely defined using our lattice-based semantics.

Finally, in a research roadmap presented in a recent review on combinatorial testing [38], Yilmaz et al. suggest the challenge "to handle evolving [combinatorial] models as they change over time". Our work starts going in this direction.

VII. Conclusion and Future Work

In this work we propose a first syntactic and semantic differencing technique for combinatorial models of test designs, and present an efficient algorithm for computing it. To enable semantic differencing, we suggest a concise and canonical representation of a model that is used as the basis for differencing. We implemented our technique in our CTD tool IBM FOCUS, and evaluated it on 42 real-world models with 107 versions, demonstrating its acceptable performance times. Our analysis reveals that real-world commits of combinatorial models tend to be large and complex in practice.

We further conducted a user study with CTD practitioners on real-world combinatorial model versions. The study showed a significant improvement of over 40 percent in the performance of less experienced practitioners when using our differencing techniques, and an improvement in confidence of expert practitioners in their comprehension of model changes.

The present paper is part of our larger research project, on comprehension and evolution of combinatorial models for test designs. In a recent paper we presented the use of visualization for comprehension of combinatorial models and test plans [34]. As part of this project, we plan to further analyze model differences and identify change patterns that commonly occur in real-world combinatorial model evolution, such as abstraction, refinement, and refactoring. Finally, we plan to investigate co-evolution of models and the test plans derived from them, and find practical and efficient ways to update a test plan following changes to the model from which it was derived.

VIII. Acknowledgments

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.

References

[1] M. Acher, P. Heymans, P. Collet, C. Quinton, P. Lahire, and P. Merle. Feature model differences. In *Advanced Inf. Sys. Eng.*, pages 629–645. IEEE Computer Society, 2012.

[2] Suplementary Material. http://smlab.cs.tau.ac.il/ctd/.

[3] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[4] K. Burroughs, A. Jain, and R. Erickson. Improved quality of protocol testing through techniques of experimental design. In *SUPERCOMM/ICC*, pages 745–752, 1994.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[6] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Trans. on Softw. Eng.*, 23(7):437–444, 1997.

[7] M. B. Cohen, M. B. Dwyer, and J. Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *ISSTA*, pages 129–139, 2007.

[8] M. B. Cohen, J. Snyder, and G. Rothermel. Testing across configurations: implications for combinatorial testing. *SIGSOFT Softw. Eng. Notes*, 31(6):1–9, 2006.

[9] J. Czerwonka. Pairwise testing in the real world: Practical extensions to test-case scenarios. In *PNSQC*, 2006.

[10] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-Based Testing in Practice. In *ICSE*, pages 285–294, 1999.

[11] E. Farchi, I. Segall, and R. Tzoref-Brill. Using projections to debug large combinatorial models. In *ICSTW*, pages 311–320, 2013.

[12] B. Fluri, M. Wursch, M. Pinzger, and H. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Trans. on Softw. Eng.*, 33(11):725–743, 2007.

[13] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler. An evaluation of combination strategies for test case selection. *Empirical Softw. Eng.*, 11(4):583–611, 2006.

[14] IBM Functional Coverage Unified Solution (IBM FOCUS). http://researcher.watson.ibm.com/researcher/view_group.php?id=1871.

[15] JDD. http://javaddlib.sourceforge.net/jdd/.

[16] Jenny website. http://burtleburtle.net/bob/math/jenny.html.

[17] D. R. Kuhn, R. N. Kacker, and Y. Lei. *Introduction to Combinatorial Testing*. Chapman & Hall/CRC, 2013.

[18] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software Fault Interactions and Implications for Software Testing. *IEEE Trans. on Softw. Eng.*, 30(6):418–421, 2004.

[19] R. Kuhn, Y. Lei, and R. Kacker. Practical Combinatorial Testing: Beyond Pairwise. *IT Professional*, 10(3):19–23, 2008.

[20] C. Lott, A. Jain, and S. Dalal. Modeling requirements for combinatorial software testing. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005.

[21] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment. In *ICCAD*, pages 6–9, 1988.

[22] S. Maoz and J. O. Ringert. A framework for relating syntactic and semantic model differences. In *MODELS*, pages 24–33. IEEE Computer Society, 2015.

[23] S. Maoz, J. O. Ringert, and B. Rumpe. ADDiff: semantic differencing for activity diagrams. In *ESEC/FSE*, pages 179–189, 2011.

[24] S. Maoz, J. O. Ringert, and B. Rumpe. CDDiff: Semantic differencing for class diagrams. In *ECOOP*, pages 230–254, 2011.

[25] S. Minato. *Graph-Based Representations of Discrete Functions*, pages 1–28. Springer US, 1996.

[26] C. Nie and H. Leung. A Survey of Combinatorial Testing. *ACM Comput. Surv.*, 43(2):11:1–11:29, 2011.

[27] Pairwise testing website. http://www.pairwise.org/tools.asp.

[28] S. Person, M. B. Dwyer, S. G. Elbaum, and C. S. Pasareanu. Differential symbolic execution. In *SIGSOFT FSE*, pages 226–237, 2008.

[29] X. Qu, M. B. Cohen, and K. M. Woolf. Combinatorial interaction regression testing: A study of test case generation and prioritization. In *ICSM*, pages 255–264, 2007.

[30] I. Segall, R. Tzoref-Brill, and E. Farchi. Using Binary Decision Diagrams for Combinatorial Test Design. In *ISSTA*, pages 254–264, 2011.

[31] K. Tai and Y. Lie. A Test Generation Strategy for Pairwise Testing. *IEEE Trans. on Softw. Eng.*, 28(1):109–111, 2002.

[32] T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *ICSE*, pages 254–264, 2009.

[33] R. Tzoref-Brill and S. Maoz. Lattice-based semantics for combinatorial model evolution. In *ATVA*, pages 276–292, 2015.

[34] R. Tzoref-Brill, P. Wojciak, and S. Maoz. Visualization of Combinatorial Models and Test Plans. In *ASE*, pages 144–154. ACM, 2016.

[35] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *TestCom*, pages 59–74, 2000.

[36] P. Wojciak and R. Tzoref-Brill. System Level Combinatorial Testing in Practice – The Concurrent Maintenance Case Study. In *ICST*, pages 103–112, 2014.

[37] Z. Xing and E. Stroulia. Differencing logical UML models. *Autom. Softw. Eng.*, 14(2):215–259, 2007.

[38] C. Yilmaz, S. Fouché, M. B. Cohen, A. A. Porter, G. Demiröz, and U. Koc. Moving forward with combinatorial interaction testing. *IEEE Computer*, 47(2):37–45, 2014.