Softw Syst Model
DOI 10.1007/s10270-016-0552-y

SPECIAL SECTION PAPER

A framework for relating syntactic and semantic model differences

Shahar Maoz[1] · **Jan Oliver Ringert**[1]

Received: 24 January 2016 / Revised: 18 July 2016 / Accepted: 23 July 2016
© Springer-Verlag Berlin Heidelberg 2016

Abstract Model differencing is an important activity in model-based development processes. Differences need to be detected, analyzed, and understood to evolve systems and explore alternatives. Two distinct approaches have been studied in the literature: syntactic differencing, which compares the concrete or abstract syntax of models, and semantic differencing, which compares models in terms of their meaning. Syntactic differencing identifies change operations that transform the syntactical representation of one model to the syntactical representation of the other. However, it does not explain their impact on the meaning of the model. Semantic model differencing is independent of syntactic changes and presents differences as elements in the semantics of one model but not the other. However, it does not reveal the syntactic changes causing these semantic differences. We define *Diffuse*, a language-independent, abstract framework, which relates syntactic change operations and semantic difference witnesses. We formalize fundamental relations of *necessary*, *exhibiting*, and *sufficient* sets of change operations and analyze their properties. We further demonstrate concrete instances of the Diffuse framework for three different popular modeling languages, namely class diagrams, activity diagrams, and feature models. The Diffuse framework provides a novel foundation for combining syntactic and semantic differencing.

Communicated by Dr. Jordi Cabot and Alexander Egyed.

✉ Shahar Maoz
maoz@cs.tau.ac.il

✉ Jan Oliver Ringert
ringert@se-rwth.de

[1] School of Computer Science, Tel Aviv University, Tel Aviv, Israel

Keywords Model differencing · Semantics · Model evolution

1 Introduction

Model differencing is an important activity in model-based development processes [2,3]. Differences between model versions need to be detected, analyzed, and understood by engineers in order to evolve systems and explore design alternatives.

Two distinct approaches to model differencing have been studied in the literature: syntactic differencing, e.g., [2,7,18,19,21,35], which compares the concrete or abstract syntax of models in terms of additions, deletions, and modifications of elements (whether reverse engineered or recorded during editing), and semantic differencing, e.g., [1,11,12,22,25,26,29,36], which compares models in terms of their meaning, e.g., instances or executions allowed only by one and not the other.

Syntactic differencing identifies (partially ordered) sets of change operations that transform the syntactical representation of one model into the syntactical representation of the other. Example change operations include inserting an activity node into an activity diagram (AD), deleting an association from a class diagram (CD), and moving a feature to be a subfeature of another feature in a feature model (FM). However, such operations do not explain the impact of the changes on the model in terms of elements removed or added to its semantics. Models with very different syntactic representations may have the same semantics; models with only few syntactic changes between them may have very different semantics. A syntactic comparison is unable to capture these differences.

Published online: 12 August 2016

Semantic model differencing is independent of syntactic changes and presents differences as elements in the semantics of one model but not the other [25]. We present background on the definitions of semantics and semantic differences that we use in this paper in Sect. 4. Example elements in the semantics of models are the object models allowed by a class diagram [29], the traces of action executions possible for an activity diagram [26], or the product configurations defined by a feature model [1,12]. The elements in the semantics of one model but not the other are called diff witnesses. However, such witnesses do not reveal the causes for the semantic difference in terms of the syntactic change operations that the engineer has applied. Awareness of these causes is important in most model evolution activities, e.g., differencing and merging, which center around syntactic change operations [3].

Despite the achievements of both, the syntactic differencing approach and the semantic differencing approach, a framework that relates them has not yet been suggested. Thus, no previous work provides the means to reason about the relations between syntactic changes on the level of individual change operations and diff witnesses.

Our objective is to support understanding the impact of change operations on the semantics of models and tracing a diff witness to syntactic changes. On the one hand, we aim to answer questions raised during the analysis of a syntactic difference given as change operations:

- Which diff witnesses are obtained when applying this set of change operations?
- Which diff witnesses are guaranteed when applying at least this set of change operations?
- Which diff witnesses will be lost when omitting these change operations?

Given two ADs, a concrete example question is: *Which traces of actions are guaranteed in the semantic difference of the two ADs when applying the change that removes node* `CloseClaim`*?*

On the other hand, we want to provide answers also to questions raised during the inspection of diff witnesses resulting from a semantic comparison of two models:

- Which syntactic change operations exhibit this diff witness?
- Which syntactic change operations guarantee this diff witness?
- Which syntactic change operations should be omitted to avoid this witness?

Given two CDs, a concrete example question is: *Which change operations should be omitted to avoid an object of class* `Manager` *managing itself?*

In this paper, we define *Diffuse*, a language-independent, abstract framework, which relates change operations and semantic diff witnesses. We formalize the relations of *necessary* change operations that a witness depends on, of *exhibiting* change operations whose application leads to a model with semantics containing the witness, and of *sufficient* change operations whose application guarantees the existence of a diff witness. The three relations of the framework relate sets of changes to diff witnesses. As the set of diff witnesses is in many cases very large or even infinite, a concrete representation of the relations is not always possible. We sketch algorithms to compute necessary, exhibiting, and sufficient sets for a given diff witness and characterize diff witnesses of the inverse relation.

We further instantiate and apply the abstract Diffuse framework to three concrete modeling languages, activity diagrams, class diagrams, and feature models, with existing, previously defined and published families of change operations from [21,33,36] and of semantic differencing operators from [1,22,26,29]. The languages used in these applications are very different in terms of their syntax, semantics, and kinds of change operations, and so demonstrate the broad applicability of the framework. Thus, the Diffuse framework provides a general foundation for combining syntactic and semantic differencing.

Our previous conference paper with the same title [32] has introduced parts of the Diffuse framework. This paper extends it by including the natural relation of *exhibiting* change operations, which was omitted from the conference version due to space limitations. We discuss how all three relations interact and provide characterizations and algorithms for each. We have also extended the theory by considering not only diff witnesses but also elements removed from the semantics of the models. We have added information on how this extension preserves properties of relations and how the algorithms can be modified to support it. The present paper introduces a grouping mechanism for change operations and examines its impact on the defined relations between change operations and diff witnesses. Related work is discussed more thoroughly than in the conference version. Finally, we motivate a concrete application of explaining change operations with witnesses.

1.1 Paper structure

We organize the remainder of this paper as follows. First, in the next section we describe three examples of changes to ADs, CDs, and FMs and their analysis. Section 3 introduces usage scenarios and discusses the challenges involved in relating syntax and semantics in the context of differencing. Section 4 presents formal background and foundations of our work. Section 5 presents the main contribution of our work, namely the Diffuse framework for relating syntactic

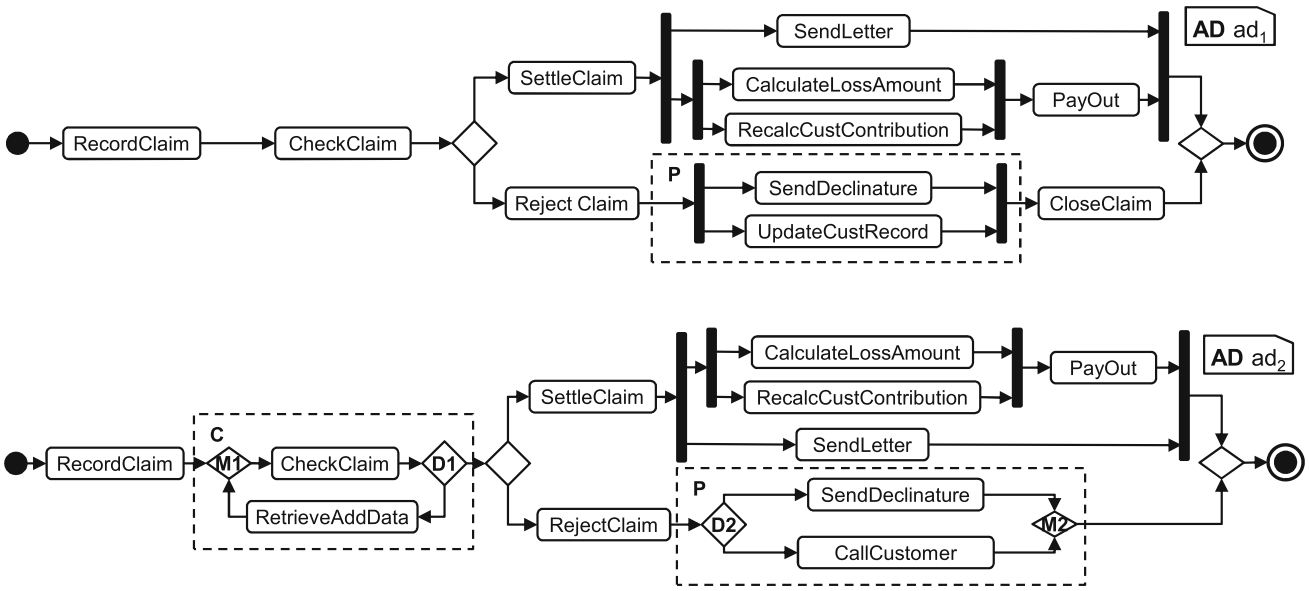

Fig. 1 Two versions of an activity for handling insurance claims (adapted from [20])

and semantic differences. Section 6 shows instantiations of Diffuse for concrete languages and difference formalizations. Section 7 presents important extensions and applications of the framework. We discuss limitations and remaining challenges in Sect. 8. Section 9 discusses related work, and Sect. 10 concludes and suggests future work directions.

2 Examples

We start with semi-formal examples of syntactic differencing, semantic differencing, and their combinations, applied to activity diagrams, class diagrams, and feature models.

2.1 Insurance company activity change

We adapt an example from [20] of an insurance company that changes its workflow for handling insurance claims. The ADs ad_1 before and ad_2 after the change are shown in Fig. 1. A syntactic analysis of the two versions reveals a set of change operations, e.g., inserting the merge and decision nodes `M1` and `D1`, inserting the action node labeled `RetrieveAddData`, and deleting the action node labeled `CloseClaim` etc.

After performing the changes, the engineer is interested in the effect of inserting action node `RetrieveAddData`. An analysis based on the Diffuse framework reveals that all executions tr_1 = <`RecordClaim, CheckClaim, RetrieveAddData, CheckClaim,..`> starting with recording and checking the claim and then retrieving additional data and checking again are semantic diff witnesses this operation is necessary for. They cannot exist without it. These execution traces are only possible in ad_2 and not in ad_1. Note that the insertion of the action node alone does not exhibit the diff witness yet. The change operation of adding fragment `C` consisting of nodes `M1` and `D1` shown in Fig. 1 is needed in addition to exhibit a diff witness with prefix tr_1.

Later, another engineer observes an execution of the activity where the claim is recorded, checked, and then rejected, and a declinature is sent, but the claim is not closed. This execution is a new behavior only possible in ad_2 and not in ad_1 (as is found by our ADDiff tool [26]). An analysis based on our new framework reveals that the trace tr_4 = <`RecordClaim, CheckClaim, RejectClaim, SendDeclinature`> is an effect of the necessary and sufficient pair of change operations deleting action nodes labeled `UpdateCustRecord` and `CloseClaim`.

The above two analyses, enabled by the Diffuse framework, are not supported by any previous work.

2.2 Class model evolution in graphical modeling framework

The Graphical Modeling Framework (GMF) is an Eclipse framework for creating graphical editors.[1] Its meta-models are formalized as Ecore models. Figure 2 shows excerpts from two consecutive versions of GMF Ecore models as CDs cd_1 and cd_2. Examples for changes from cd_1 to cd_2 are the addition of the abstract class `CustomAttributeOwner` as a superclass of `RealFigure` and `CustomClass`, the move of the composition between `CustomClass`

[1] Eclipse Graphical Modeling Project. http://www.eclipse.org/modeling/gmp/

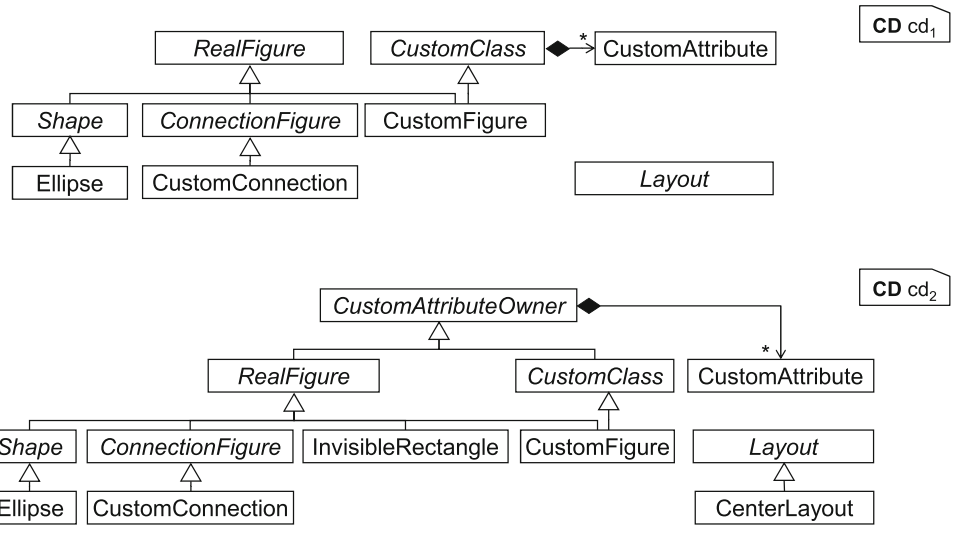

Fig. 2 Excerpts of two consecutive versions of the Eclipse GMFgraph metamodel cd_1 committed 2012-04-29 and cd_2 committed 2012-05-14

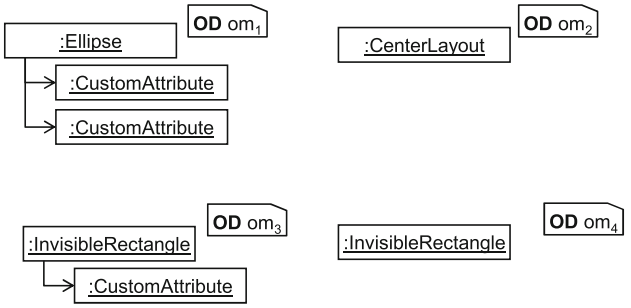

Fig. 3 Object model instances of class diagram cd_2 depicted as object diagrams

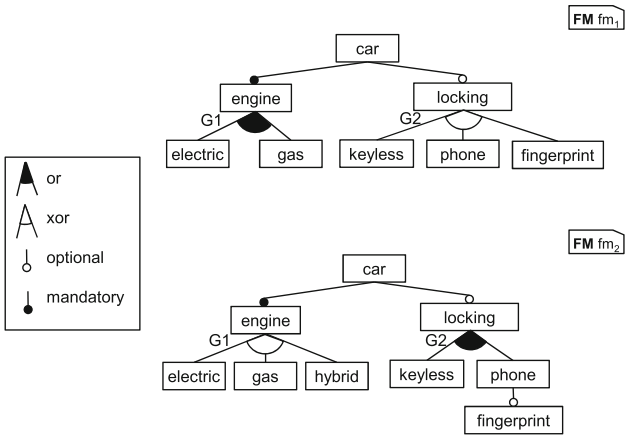

Fig. 4 Two feature models shown as feature diagrams describing valid configurations of a car inspired by examples from [8]

and `CustomAttribute` to be owned by the new class `CustomAttributeOwner`, and the addition of the class `CenterLayout`.

During semantic differencing of the two CDs, e.g., using our CDDiff tool [29], an engineer wonders about an `Ellipse` owning instances of `CustomAttribute` (see Fig. 3, om_1), which was not possible in cd_1. She investigates the change using the Diffuse framework and finds that the addition of the class `CustomAttributeOwner`, the inheritance to `RealFigure`, and the moving of the composition with `CustomAttribute`, exhibit this witness. These changes are also sufficient for the witness, i.e., they guarantee the existence of the witness independently of the other changes from cd_1 to cd_2.

Another engineer uses the Diffuse framework to check whether the addition of class `CenterLayout` is necessary for any semantic difference. The change indeed allows new instances of cd_2 not possible in cd_1 with objects of type `CenterLayout` (see Fig. 3, om_2). The change is necessary and by itself already sufficient for the witness om_2.

Again, the above analyses, enabled by the Diffuse framework, are not supported by any previous work.

2.3 Car feature configurations with feature models

Consider a car configuration system inspired by examples from [8], whose valid configurations are formally specified by the FMs shown in Fig. 4. The FMs express alternatives for the engine and locking system of the car. In fm_1, the choice between the keyless entry locking system (feature `keyless`), phone-operated locking system (feature `phone`), and fingerprint-secured locking system (feature `fingerprint`), is an exclusive choice (see `xor` feature group `G2` in fm_1 from Fig. 4), i.e., exactly one of the options has to be chosen.

To make the configuration process clearer, an engineering team has decided to add the explicit feature `hybrid` in fm_2 for the engine of the car instead of selecting both `electric` and `gas` in fm_1. The engine selection is now an exclusive alternative in fm_2. Also, the feature `fingerprint` is a subfeature of `phone` in fm_2 and the choice of a locking system is no longer exclusive.

While browsing configurations in the semantic difference of the two FMs, an engineer detects the configuration c_1 = {car, engine, electric, locking, phone, fingerprint} and is interested in the change operations this witness results from. She finds that both, moving fingerprint below phone or making feature group G2 a non-exclusive choice, are exhibiting the witness.

As a routine check, one of the engineers checks whether all change operations contribute diff witnesses. This is not the case for the operation changing feature group G1 to an exclusive alternative. It turns out that this change is a refinement, which removes but does not add valid configurations. All other change operations participate in minimal sets of changes sufficient for diff witnesses.

These examples, with activity diagrams, class diagrams, and feature models, illustrate the relations between syntactic change operations and semantic witnesses that the Diffuse framework supports and demonstrate how their explication provides explanations for syntactic causes and semantic effects.

3 Usage scenarios and the challenge of relating semantic and syntactic differences

We first describe usage scenarios of applying the Diffuse framework for relating semantic and syntactic differences. Then, we discuss the main challenges in defining relations between the two views on differences.

3.1 Usage scenarios

It is important to note that the Diffuse framework focuses on comprehension and not on manipulation of model differences. Comprehension of model differences by engineers is important in many model maintenance tasks. The two main usage scenarios we motivated with example questions in the introduction and concretized in Sect. 2 are (1) understanding the impact of a subset of change operations on the semantics of the two models and (2) tracing a diff witness to change operations.

To support understanding the impact of change operations on the semantics of the model, the Diffuse framework defines relations of different strengths. Typical comprehension tasks supported by these relations are: Given a subset of change operations

- Which diff witnesses are obtained when applying this set of change operations?
- Which diff witnesses are guaranteed when applying at least this set of change operations?
- Which diff witnesses will be lost when omitting these change operations?

Given two FMs, a concrete example question is: *Which products are in the semantic difference of the two FMs when changing this feature group to an exclusive alternative?*

Given two ADs, a concrete example question is: *Which traces of actions are guaranteed in the semantic difference of the two ADs when applying the change that removes node* `CloseClaim`*?*

Given two CDs, a concrete example question is: *Which object models will not appear in the semantic difference of the two CDs when not adding the generalization between* `Manager` *and* `Person`*?*

We formalize these relations in Sect. 5 and provide generic characterizations and algorithms to compute witnesses answering the above questions. These formalizations may serve as foundations for tools for analyzing change operations.

Typical comprehension tasks for tracing a diff witness to change operations are: Given a witness

- Which syntactic change operations exhibit this diff witness?
- Which syntactic change operations guarantee this diff witness?
- Which syntactic change operations should be omitted to avoid this witness?

Given two ADs, a concrete example question is: *Which operations applied to the AD exhibit the trace* `<RecordClaim, CheckClaim, RejectClaim, CallCustomer>`*?*

Given two FMs, a concrete example question is: *Which of these change operations guarantee the diff witness* `{car, engine, hybrid, locking, keyless, phone}`*?*

Given two CDs, a concrete example question is: *Which change operations should be omitted to avoid an object of class* `Manager` *managing itself?*

Answers to these questions are provided by the relations defined in the Diffuse framework. Concrete implementations for modeling languages may help guide the engineer through syntactic changes starting from differences observed in the models' semantics.

The Diffuse framework complements existing differencing approaches by allowing engineers to investigate the relation between syntactic and semantic differences.

3.2 On the challenge of relating syntactic and semantic differences

Defining a relation between syntactic and semantic model differences is not trivial. First, the domains at hand are very different. While the syntactic domain of a modeling language might be expressed as a graph structure or a syntax tree, its semantic domain can contain sets of very different elements

representing action traces, object models, or feature configurations. Second, the same modeling language syntax might be mapped to different semantic domains depending on the purpose of the mapping. While the syntax-based representation of a single model is finite, the semantics of a single model might contain infinitely many elements. Finally, the rich syntax of modeling languages usually allows models with very different syntax to express the same semantics.

On top of these differences between the syntactic domain and the semantic domain of a modeling language, we are interested in relating their changes. This adds another dimension to the challenge. The same syntactic change may be applied to different models with very different semantic effects, e.g., the deletion of an action node from an AD might be a refinement if the node was in an alternative fragment or it leads to completely different semantics if the node exclusively followed the initial node. Moreover, typically, the syntactic difference between versions of models consists of more than one change operation. In our context, these change operations have to be considered together because they may be semantically dependent, e.g., first adding a class to a CD and then making it abstract reverts the impact of the first change on the semantics of the CD in terms of added object models.

To address these challenges we first introduce general characterizations of modeling languages, syntactic difference, and semantic difference and then relate differences in an abstract, generic way. To illustrate properties and demonstrate the applicability of the Diffuse framework, we show how to instantiate it for concrete modeling languages on the example of ADs, CDs, and FMs.

4 Preliminaries

We define a general setup for connecting syntactic and semantic differences of modeling languages. The definitions of relations we identified apply to modeling languages with set-based semantics, and syntactic differences characterized as syntactic change operations. We now review these general properties of modeling languages, syntactic differences, and semantic differences.

4.1 Modeling language

A modeling language $\mathcal{L} = \langle \mathcal{M}, \mathcal{S}, sem \rangle$ is a tuple where $\mathcal{M}$ is the set of all syntactically correct (i.e., well-formed) models according to some syntax definition, $\mathcal{S}$ is a semantic domain, and $sem : \mathcal{M} \to \wp(\mathcal{S})$ defines the meaning of a model $m \in \mathcal{M}$ by mapping the model to a set of elements of $\mathcal{S}$ (see [15]). The semantic domain is typically a well-understood model that allows for formal analyses. The semantic mapping relates well-formed models to elements of the semantic domain to define their meaning.

As an example, the syntax of ADs contains nodes and edges. Examples of nodes are initial nodes, shown in the example ADs in Fig. 1 as filled circles, and action nodes, shown as boxes with rounded edges and action names. The set of well-formed ADs contains the two ADs from Fig. 1, but it does not contain any AD with an initial node that has incoming edges. A semantic domain of ADs is the set of traces consisting of action names in the order these can be executed in the AD. The semantic mapping for this semantic domain of ADs relates a diagram to the set of traces describing all valid executions of the activity, e.g., the first action name appearing in the trace is the label of an action node preceded by an initial node. As a concrete example, all traces in the semantics of both ADs shown in Fig. 1 start with the action names `<RecordClaim, CheckClaim,... >` because these actions are always executed first.

Note that the syntax and semantics of modeling languages are usually expressed by very different structures. For most modeling languages, syntactically very different models $m_1 \neq m_2 \in \mathcal{M}$ may have the same semantics $sem(m_1) = sem(m_2) \in \wp(\mathcal{S})$.

4.2 Syntactic difference

Given two models $m_1, m_2 \in \mathcal{M}$ we describe their syntactic difference in terms of partially ordered change operations $\mathcal{C}$ as $\Delta(m_1, m_2) = \{c_1, .., c_n\} \subseteq \mathcal{C}$ where the application of the change operations $\{c_1, .., c_n\}$ to the first model leads to the second model (see, e.g., [2,7,20,21,35]). Change operations are not necessarily independent [2,7,20,35], and their dependencies are typically represented as a partial application order (reflexive, transitive, antisymmetric). It is important to note that the partial order is defined for change operations in the syntactic difference $\Delta(m_1, m_2)$ and in general not for the set of all possible change operations $\mathcal{C}$. In some cases, change operation dependency is defined also for partial applications [20].

We denote the application of change operations (according to their given order) using the operator $\oplus : \mathcal{M} \times \wp(\mathcal{C}) \rightharpoonup \mathcal{M}$, e.g., $m_1 \oplus \Delta(m_1, m_2) = m_2$. The operator is partial because some applications of change operations are not possible due to application dependencies [14,19,35] or their application does not produce well-formed models in $\mathcal{M}$. To denote that the application of change operations $C \subseteq \Delta(m_1, m_2)$ to a model $m_1 \in \mathcal{M}$ leads to a well-formed model, we write $m_1 \oplus C \in \mathcal{M}$.

Note that for two models $m_1, m_2 \in \mathcal{M}$, the syntactic difference in terms of change operations $\Delta(m_1, m_2)$ is not necessarily unique. However, when we relate syntactic and semantic differences in Sect. 5, we refer to one set of change

operations as recorded by a model editor or produced by syntactic differencing operators [2,7,20,21,35].

Some interesting relations can be defined with respect to minimal and maximal sets based on standard definitions of the partial order of set inclusion. Given a set of sets $\mathcal{X}$ characterizing a property, a set $C_{\max} \in \mathcal{X}$ is maximal wrt. this property iff $\forall C \in \mathcal{X} : C_{\max} \subseteq C \Rightarrow C = C_{\max}$ and a set $C_{\min} \in \mathcal{X}$ is minimal iff $\forall C \in \mathcal{X} : C \subseteq C_{\min} \Rightarrow C = C_{\min}$. In most cases throughout the paper $\mathcal{X}$ will consist of subsets of the syntactic difference of two models $\Delta(m_1, m_2)$.

4.3 Semantic difference

The semantic difference of two models $m_1, m_2 \in \mathcal{M}$ is a set of diff witnesses in the semantic domain $\mathcal{S}$. Diff witnesses are elements added to the semantics of the model between the two versions. The semantic differencing operator is defined as $\delta(m_1, m_2) = sem(m_2) \setminus sem(m_1)$ [25].

Note that δ is not symmetric. The semantic difference of a model and itself is empty $\delta(m, m) = \emptyset$ and no witnesses are in the intersection of the semantic difference and its reverse application $\delta(m_1, m_2) \cap \delta(m_2, m_1) = \emptyset$.

If all elements in the semantics of m_2 are included in the semantics of m_1, i.e., $\delta(m_1, m_2) = \emptyset$, we call m_2 a refinement of m_1. The semantics of two models are equivalent if and only if $\delta(m_1, m_2) = \delta(m_2, m_1) = \emptyset$.

5 Relating change operations and diff witnesses

We are now ready to introduce the Diffuse framework, which is the main contribution of this paper. First, we formally define the fundamental relations of necessary, exhibiting, and sufficient syntactic changes for semantic diff witnesses in Sect. 5.1. Second, we present language-independent algorithms for computing necessary, exhibiting, and sufficient sets of change operations in Sect. 5.2. Finally, complementary to the algorithms, we present characterizations of diff witnesses for necessary, exhibiting, and sufficient sets in Sect. 5.3.

Note that for two models $m_1, m_2 \in \mathcal{M}$ the Diffuse framework does not relate arbitrary change operations to semantic diff witnesses. It relates subsets of change operations from the syntactic difference $\Delta(m_1, m_2)$ to semantic diff witnesses from $\delta(m_1, m_2)$.

5.1 Necessary, exhibiting, and sufficient change operations

We define three relations between sets of syntactic change operations and semantic diff witnesses. The following definitions express when sets of change operations are necessary for a witness, are exhibiting a witness, or are sufficient for a witness.

5.1.1 Necessary change operations

For two models $m_1, m_2 \in \mathcal{M}$ a set of change operations from the set of all change operations in the syntactic difference $C_{nec} \subseteq \Delta(m_1, m_2)$ is necessary for the existence of a diff witness $w \in \delta(m_1, m_2)$ iff for all applications of change operations $C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M}$, the existence of the diff witness in $sem(m_1 \oplus C')$ depends on the containment of C_{nec} in C'. Formally:

Definition 1 *(Necessary change operations)* For models $m_1, m_2 \in \mathcal{M}$, a set of necessary change operations for a diff witness $w \in \delta(m_1, m_2)$ is a set $C_{nec} \subseteq \Delta(m_1, m_2)$ such that $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : w \in sem(m_1 \oplus C') \Rightarrow C_{nec} \subseteq C'$.

We denote the set of all sets of necessary change operations for models $m_1, m_2 \in \mathcal{M}$ and a witness $w \in \delta(m_1, m_2)$ by $nec(m_1, m_2, w)$.

As a simple example, consider the change operations from cd_1 to cd_2 shown in Fig. 2. The addition of class `CenterLayout` is in the set of necessary change operations for the witness om_2 consisting of an object of this class. In contrast, the addition of class `CustomAttributeOwner` is not in the set of necessary change operations for this witness.

Note that there may be witnesses without necessary change operations. As an example, consider the changes from fm_1 to fm_2 shown in Fig. 4. The diff witness c_1 = `{car, engine, electric, locking, phone, fingerprint}` has no necessary change operations. This witness can be obtained via two independent applications of change operations, which are both exhibiting the witness (see Sect. 2.3).

We now explore some interesting properties of the relation from Definition 1. First, we note that any subset of a set of necessary change operations is also a set of necessary change operations.

Theorem 1 (Subsets necessary sets) *Given a set of necessary change operations $C_{nec} \in nec(m_1, m_2, w)$, all subsets $C' \subseteq C_{nec}$ are also sets of necessary change operations for w.*

Theorem 1 follows directly from Definition 1 because the containment of C_{nec} implies the containment of all its subsets.

Second, for all models and witnesses the empty set is the unique minimal necessary set (see Definition 1).

Third, the maximal set of necessary change operations for a witness is unique.

Theorem 2 (Max necessary set unique) *The maximal necessary set of change operations* $C_{nec} \in nec(m_1, m_2, w)$ *is unique.*

Proof Assume C_{nec} and C'_{nec} both maximal members of $nec(m_1, m_2, w)$. From Definition 1, it follows that $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : w \in sem(m_1 \oplus C') \Rightarrow C_{nec} \subseteq C' \wedge C'_{nec} \subseteq C'$, and thus, $(C_{nec} \cup C'_{nec})$ is a necessary set. From the definition of maximal, we get $C_{nec}, C'_{nec} \subseteq (C_{nec} \cup C'_{nec}) \Rightarrow C_{nec} = (C_{nec} \cup C'_{nec}) = C'_{nec}$. □

5.1.2 Exhibiting change operations

A set of change operations can be necessary for a witness even though the application of these operations may not lead to a model with the witness in its semantics. We thus define the stronger notion of exhibiting a witness. This is the most intuitive relation between change operations and witnesses, meaning that the witness is in the semantics of the model after application of the change operations.

Definition 2 *(Exhibiting change operations)* For models $m_1, m_2 \in \mathcal{M}$, a set of exhibiting change operations for a diff witness $w \in \delta(m_1, m_2)$ is a set $C_{exh} \subseteq \Delta(m_1, m_2)$ such that $(m_1 \oplus C_{exh}) \in \mathcal{M}$ and $w \in sem(m_1 \oplus C_{exh})$.

We denote the set of all sets of exhibiting change operations for models $m_1, m_2 \in \mathcal{M}$ and a witness $w \in \delta(m_1, m_2)$ by $exh(m_1, m_2, w)$.

As a simple example, consider the change operations from fm_1 to fm_2 shown in Fig. 4. The singleton set consisting of the move of feature `phone` below feature `fingerprint` is a set of exhibiting change operations for the witness c_1 containing both features `phone` and `fingerprint`. In contrast, the set consisting of the change operation of adding feature `hybrid` is not in the set of exhibiting change operations for this witness.

Interestingly, a minimal set of exhibiting change operations for a witness w is not necessarily unique, and exhibiting sets can be disjoint. Continuing the above example of fm_1, fm_2, and the witness c_1, another set of exhibiting change operations for this witness is the singleton set of changing feature group `G2` from an exclusive alternative to a non-exclusive alternative.

The maximal set of exhibiting change operations for all witnesses $w \in \delta(m_1, m_2)$ is always $\Delta(m_1, m_2)$.

A natural relation between necessary and exhibiting sets of change operations is that all sets of necessary change operations are contained in every set of exhibiting change operations.

Theorem 3 (Exhibiting contains necessary) *For every witness, all sets of necessary change operations are contained in every set of exhibiting change operations:*

$$\forall w \in \delta(m_1, m_2), \; C_{nec} \in nec(m_1, m_2, w), C_{exh} \in exh(m_1, m_2, w) : \; C_{nec} \subseteq C_{exh}$$

Proof Follows from Definition 1 and Definition 2 because every exhibiting set C_{exh} has a well-formed application to m_1 and satisfies the antecedent of implication $w \in sem(m_1 \oplus C') \Rightarrow C_{nec} \subseteq C'$. □

Interestingly, neither subsets nor extensions of sets of exhibiting change operations are guaranteed to be sets of exhibiting change operations. As an example for the extension of a set of exhibiting change operations, consider the changes from cd_3 to cd_4 shown in Fig. 8 and the witness om_6 shown in Fig. 9 presented in Sect. 6.2. Introducing a generalization relation (change 1 in Fig. 9) exhibits the witness om_6, but extending the set of change operations with adding a new class and moving an association (changes 2 and 3 in Fig. 9) leads to a set that does not exhibit the witness.

5.1.3 Sufficient change operations

Finally, we are interested not only in sets of necessary or exhibiting change operations but also in sets of change operations that are sufficient to produce a given witness in the semantic difference. In contrast to exhibiting a witness, sufficiency is robust against additional change operations, i.e., the application of a set of sufficient change operations $C_{suff} \subseteq \Delta(m_1, m_2)$ guarantees the existence of the witness regardless of the application of additional changes from $\Delta(m_1, m_2)$. Formally:

Definition 3 *(Sufficient change operations)* For models m_1, $m_2 \in \mathcal{M}$, a set of sufficient change operations for a diff witness $w \in \delta(m_1, m_2)$ is a set $C_{suff} \subseteq \Delta(m_1, m_2)$ such that $(m_1 \oplus C_{suff}) \in \mathcal{M}$ and $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : C_{suff} \subseteq C' \Rightarrow w \in sem(m_1 \oplus C')$.

We denote the set of all sets of sufficient change operations for models $m_1, m_2 \in \mathcal{M}$ and a witness $w \in \delta(m_1, m_2)$ by $suff(m_1, m_2, w)$.

As a simple example for a sufficient set of change operations, consider the deletions of the action nodes labeled `UpdateCustRecord` and `CloseClaim` from Fig. 1. The two operations together are sufficient for the diff witness tr_4 from the example in Sect. 2.1. Each operation alone is not sufficient for this witness.

The definition of sufficiency contains the existence of the witness in the semantics of the model after the application of the change operations. Following Definition 2, every sufficient set is thus also an exhibiting set of change operations. It is very important to note that sufficiency is indeed stronger than exhibiting the diff witness after application of a set of

change operations. As an example, consider cd_3 and cd_4 shown in Fig. 8 and the diff witness om_6 shown in Fig. 9. The diff witness om_6 is in the semantics of cd_3 after adding the generalization between classes `Manager` and `Employee`. However, this change operation is not sufficient to guarantee the existence of the diff witness. The diff witness is lost when further adding class `Person` and moving the association end `addressOwner` to `Person`.

Interestingly, a minimal set of sufficient change operations for a witness w is not necessarily unique, and sufficient sets can be disjoint as in the example of FMs for witness c_1 (see Sect. 2.3). However, adding change operations to a set of sufficient change operations for w results in a set of sufficient change operations for w.

Theorem 4 (Well-formed extensions of sufficient are sufficient) *Given a set of sufficient change operations $C_{suff} \in suff(m_1, m_2, w)$ all its extensions C' with $(m_1 \oplus C') \in \mathcal{M}$ are sets of sufficient change operations for w, i.e., $\forall C_{suff} \subseteq C' \subseteq \Delta(m_1, m_2) : (m_1 \oplus C') \in \mathcal{M} \Rightarrow C' \in suff(m_1, m_2, w)$.*

Theorem 4 follows directly from Definition 3. As a corollary, note that the maximal set $C' = \Delta(m_1, m_2)$ is sufficient for every witness. Due to the extensibility of sets of sufficient change operations up to $\Delta(m_1, m_2)$, we consider the minimal sufficient sets for a witness the most informative.

5.1.4 Additional properties of the three relations

The relations between witnesses and their necessary and sufficient sets of change operations are dual, in the sense that the existence of a witness implies the application of its necessary change operations, while the application of the sufficient change operations implies the existence of the witness.

Similar to the containment of necessary sets of change operations in exhibiting sets of change operations, all sets of necessary change operations for w are also contained in every set of sufficient change operations for w.

Theorem 5 (Sufficient contains necessary) *For every witness, all sets of necessary change operations are contained in every set of sufficient change operations:*

$$\forall w \in \delta(m_1, m_2),$$
$$C_{nec} \in nec(m_1, m_2, w), C_{suff} \in suff(m_1, m_2, w) :$$
$$C_{nec} \subseteq C_{suff}$$

Proof Follows from Definition 1 and 3 because every sufficient set C_{suff} has a well-formed application to m_1 and satisfies the antecedent of implication $w \in sem(m_1 \oplus C') \Rightarrow C_{nec} \subseteq C'$. □

Furthermore, all exhibiting sets of change operations are (not necessarily strictly) contained in at least one sufficient set of change operations.

If the minimal sufficient set and the maximal necessary set of change operations coincide, this set is also the unique minimal exhibiting set of change operations.

Corollary 1 (Coincidence) *For every witness with minimal necessary set C_{nec} and maximal sufficient set $C_{suff} = C_{nec}$, the set C_{suff} is also the unique minimal exhibiting set of the witness.*

Corollary 1 follows from the containment of the maximal necessary set in every exhibiting set (Theorem 3) and the observation that every sufficient set is also an exhibiting set.

The intersection of all exhibiting sets for a witness w is the unique maximal necessary set of change operations of this w.

Theorem 6 (Intersection of exhibiting is maximal necessary set) *For all models $m_1, m_2 \in \mathcal{M}$ and $w \in \delta(m_1, m_2)$:*

$$\bigcap_{C_{exh} \in exh(m_1, m_2, w)} C_{exh} = \max_{C_{nec} \in nec(m_1, m_2, w)} C_{nec}.$$

Proof Recall Theorem 2 that the maximal necessary is unique. "$\supseteq$": follows from Theorem 3, "$\subseteq$": the intersection on the left side satisfies Definition 1 and is thus contained in $nec(m_1, m_2, w)$.

□

Note that the intersection of all sufficient sets contains but does not necessarily coincide with the maximal necessary set. As an example, consider the changes from cd_3 to cd_4 shown in Fig. 8. The witness om_6 consisting of a manager with one address (see right side of Fig. 9) has the addition of the generalization between employee and manager (change operation 1, Fig. 9) as the maximal necessary set of change operations, but its minimal sufficient set contains also the creation of class `Person` (change operation 2), the generalization between `Person` and `Employee` (change operation 4), and the move of the association end of `addressOwner` (change operation 3).

The relation between sets of syntactic change operations from $\Delta(cd_3, cd_4)$ and the witness $om_6 \in \delta(cd_3, cd_4)$ is illustrated in Fig. 5. Only sets that lead to a well-formed model after application to cd_3 are shown. The singleton set of creating the generalization between classes `Manager` and `Employee` (change operation 1) is a maximal necessary and minimal exhibiting, but not sufficient set of change operations. All sets including the generalization but not the move of the association end `addressOwner` (change operation 3) are exhibiting sets of change operations. Finally, the witness has two sufficient (and thus exhibiting) sets of change operations as shown in Fig. 5.

Note that exhibiting or sufficient sets of change operations for a witness are never empty sets, while necessary sets of change operations might be empty.

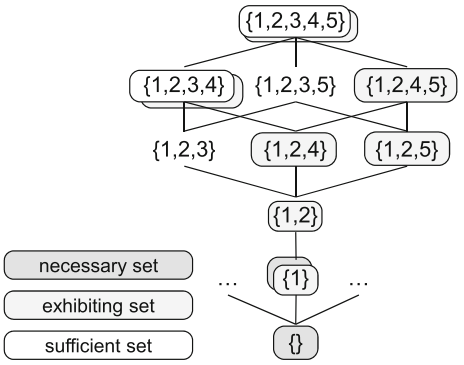

Fig. 5 Illustration of sets of syntactic change operations from $\Delta(cd_3, cd_4)$ and their relation to witness $om_6 \in \delta(cd_3, cd_4)$ shown in Fig. 9. Change operations numbers are taken from Fig. 9. Shaded boxes indicate the relation between each set of changes and the witness, whether necessary, exhibiting, or sufficient. Lines between sets represent set inclusion

5.2 Computation of necessary, exhibiting, and sufficient sets

We present three algorithms. Given two models and a diff witness as input, the first algorithm computes the maximal necessary set of change operations, the second computes all exhibiting sets of change operations, and the third computes all minimal sufficient sets of change operations. Note that we decided to compute the most informative necessary and sufficient sets, i.e., we compute the maximal necessary set because all of its subsets are also necessary sets (Theorem 2), and we compute minimal sufficient sets because all of its extensions are sufficient sets (Theorem 4). For sets of exhibiting change operations, a similar reduction does not work because neither extensions nor subsets are guaranteed to be exhibiting sets of change operations.

We give the three algorithms in pseudo-code. All three algorithms iterate over sets of change operations from the syntactic difference of the models. Basic operations used by the algorithms are the application of change operations (operator $\oplus$), checking well-formedness ($m \in \mathcal{M}$), checking membership of a witness in the semantics of a model ($w \in sem(m)$), and standard set manipulation.

The time complexity of the three algorithms depends on the operations listed above with time complexities dependent on the concrete modeling language, the change operation framework, and the modeling language semantics definition. The time complexity also depends on the size of the syntactic difference, i.e., the number of change operations in $\Delta(m_1, m_2)$.

5.2.1 Computing the maximal necessary set

Algorithm 1 computes the maximal necessary set of change operations *maxNec* for a given witness $w \in \delta(m_1, m_2)$ following Theorem 6. Intuitively, it computes the intersection of all sets of change operations that yield well-formed models with w in their semantics. The possibly empty intersection is a necessary set and satisfies Definition 1 by construction, because Algorithm 1 iterates over all C' from Definition 1.

Algorithm 1 Compute max necessary set for $w \in \delta(m_1, m_2)$

```
1: define maxNec as set
2: maxNec ← Δ(m1, m2)
3: for all C ⊆ Δ(m1, m2) do
4:    m ← m1 ⊕ C
5:    if m ∈ M ∧ w ∈ sem(m) then
6:       maxNec ← maxNec ∩ C
7:    end if
8: end for
9: return maxNec
```

The algorithm starts with the set of all change operations (l. 2), iterates over all subsets of the set of change operations (l. 3), and checks whether the application of change operations yields a well-formed model with w in its semantics (l. 5). If the model has w in its semantics, the set *maxNec* is updated to be the intersection with the applied change operations (l. 6).

As an example, given cd_1, cd_2, and om_1, Algorithm 1 computes the maximal necessary set of change operations consisting of adding class `CustomAttributeOwner`, making it a superclass of `RealFigure`, and moving the association end `attribOwner` to `CustomAttributeOwner`.

As another example, given ad_1, ad_2, and tr_4, Algorithm 1 computes the maximal necessary set of change operations consisting of only deleting the action node labeled `CloseClaim`.

The time complexity of Algorithm 1 is exponential in the size of the syntactic difference of the models m_1 and m_2 because the loop in l. 3 iterates over all subsets of $\Delta(m_1, m_2)$. For the examples from Sect. 2, these sizes are $|\Delta(ad_1, ad_2)| = 7$, $|\Delta(cd_1, cd_2)| = 9$, and $|\Delta(fm_1, fm_2)| = 5$.

5.2.2 Computing exhibiting sets

Algorithm 2 computes the set of all sets of change operations *exh* that are exhibiting a given witness $w \in \delta(m_1, m_2)$. The algorithm iterates over all subsets $C \subseteq \Delta(m_1, m_2)$ of the syntactic difference and includes a set in *exh* if applied to m_1 the witness w is in the semantics of the resulting model.

As an example, given cd_1, cd_2, and om_2, Algorithm 2 computes the set of all subsets of change operations $\Delta(cd_1, cd_2)$ that contain at least the change operation adding class `CenterLayout`. As another example, given fm_1, fm_2, and c_1 = `{car, engine, electric, locking, phone, fingerprint}` the algorithm returns all subsets of $\Delta(cd_1, cd_2)$ that contain at least the change operation

Algorithm 2 Compute exhibiting sets for $w \in \delta(m_1, m_2)$

```
1: define exh as set of sets
2: exh ← ∅
3: for all C ⊆ Δ(m1, m2) do
4:   m ← m1 ⊕ C
5:   if m ∈ M ∧ w ∈ sem(m) then
6:     add C to exh
7:   end if
8: end for
9: return exh
```

moving `fingerprint` below feature `phone` or making feature group `G2` a non-exclusive choice.

The time complexity of Algorithm 2 is exponential in the size of the syntactic difference of the models m_1 and m_2 because the loop in l. 3 iterates over all subsets of $\Delta(m_1, m_2)$.

5.2.3 Computing all minimal sufficient sets

Algorithm 3 computes the set of all minimal sufficient sets *minSuff* for a given witness $w \in \delta(m_1, m_2)$. Intuitively for every candidate set of change operations, the algorithm checks whether all extension of the set that produce a well-formed model m have $w \in sem(m)$. If this is the case for all extensions, the candidate set is a sufficient set according to Definition 3. The algorithm finds minimal sufficient sets first by iterating over the candidate sets of change operations in ascending order of their size (l. 5). Two sets of the same size are either equivalent or incomparable with the partial order of set inclusion. Candidate sets that are not minimal, i.e., contain a minimal sufficient set, are filtered out in l. 6.

Algorithm 3 Compute min sufficient sets for $w \in \delta(m_1, m_2)$

```
1: define minSuff as set of sets
2: define allExtW as Boolean
3: minSuff ← ∅
4: for all C ⊆ Δ(m1, m2) for growing size |C| do
5:   m ← m1 ⊕ C
6:   if m ∈ M ∧ w ∈ sem(m) ∧ ∄C' ∈ minSuff : C' ⊆ C then
7:     allExtW ← true
8:     for all C ⊆ C' ⊆ Δ(m1, m2) do
9:       m ← m1 ⊕ C'
10:      if m ∈ M ∧ w ∉ sem(m) then
11:        allExtW ← false
12:      end if
13:    end for
14:    if allExtW then
15:      add C to minSuff
16:    end if
17:  end if
18: end for
19: return minSuff
```

As an example, given cd_1, cd_2, and om_1, Algorithm 3 computes a minimal sufficient set of change operations consisting of adding class `CustomAttributeOwner`, making it a superclass of `RealFigure`, and moving the association end `attribOwner` to `CustomAttributeOwner`.

As another example, given ad_1, ad_2, and tr_4, Algorithm 3 computes two minimal sufficient sets of change operations: (1) delete `CloseClaim` and `UpdateCustRecord` and (2) delete `CloseClaim` and convert fragment `P` from parallel to alternative.

The time complexity of Algorithm 3 is exponential in the size of the syntactic difference of the models m_1 and m_2 because the loop in l. 5 and the nested loop in l. 8 both iterate over all subsets of $\Delta(m_1, m_2)$.

5.3 Characterization of diff witnesses

Given two models $m_1, m_2 \in \mathcal{M}$, the Diffuse framework relates diff witnesses from $\delta(m_1, m_2)$ to necessary, exhibiting, and sufficient sets of change operations from $\Delta(m_1, m_2)$. The set of diff witnesses compared to the set of change operations is typically very large or even infinite, and it is thus impractical to enumerate all diff witnesses. It is, however, possible to express the diff witnesses that a set of change operations is necessary for, is exhibiting, or is sufficient for, using basic set operations and the semantic mapping $sem : \mathcal{M} \to \mathcal{S}$.

The diff witnesses a set $C \subseteq \Delta(m_1, m_2)$ is necessary for are:

$$\delta(m_1, m_2) \setminus \left(\bigcup_{\substack{C \not\subseteq C' \subseteq \Delta(m_1, m_2) \\ m = m_1 \oplus C' \in \mathcal{M}}} sem(m) \right). \tag{1}$$

Given a set $C \subseteq \Delta(m_1, m_2)$, starting from the set of all diff witnesses $\delta(m_1, m_2)$, the characterization in Eq. (1) removes all elements in the semantics of models that are obtained from change operations without containing all change operations in C. The remaining witnesses are those that can only be obtained by applying at least all change operations in C and C is thus one of their necessary sets of change operations.

The diff witnesses a set $C \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C) \in \mathcal{M}$ is exhibiting are:

$$\delta(m_1, m_2) \cap sem(m_1 \oplus C). \tag{2}$$

The diff witnesses $C \subseteq \Delta(m_1, m_2)$ exhibits are all diff witnesses that are also in the semantics of $m_1 \oplus C$. The intersection with diff witnesses $\delta(m_1, m_2)$ is necessary because $sem(m_1 \oplus C)$ may contain elements not in $sem(m_2)$, i.e., elements that are not diff witnesses.

The diff witnesses a set $C \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C) \in \mathcal{M}$ is sufficient for are:

Algorithm 4 Compute witnesses for which $C \subseteq \Delta(m_1, m_2)$ is necessary set

```
1: define witNec as set
2: witNec ← sem(m2) \ sem(m1)
3: for all C' s.t. C ⊈ C' ⊆ Δ(m1, m2) do
4:    m ← m1 ⊕ C'
5:    if m ∈ M then
6:       witNec ← witNec \ sem(m)
7:    end if
8: end for
9: return witNec
```

$$\delta(m_1, m_2) \cap \left(\bigcap_{\substack{C \subseteq C' \subseteq \Delta(m_1, m_2) \\ m = m_1 \oplus C' \in \mathcal{M}}} sem(m) \right). \tag{3}$$

Given a set $C \subseteq \Delta(m_1, m_2)$, starting from the set of all diff witnesses $\delta(m_1, m_2)$, the characterization in Eq. (3) retains all witnesses that are contained in the semantics of all models after applying at least the change operations in C. The remaining witnesses are thus contained in the semantics after applying an extension of C; C is thus one of their sufficient sets of change operations.

The three above formalizations can be employed naively as an extension of existing semantic differencing approaches based on constraint solving, where union and intersection translate to disjunction and conjunction.

It is also possible to formulate algorithms for the witness computation similar to the algorithms in Sect. 5.2. Again, basic operations used by the algorithms are the application of change operations (operator $\oplus$) and checking well-formedness ($m \in \mathcal{M}$). In contrast to the algorithms in Sect. 5.2, the algorithms for computing witnesses now require computing the semantics of a model ($sem(m)$) instead of only checking membership, and manipulation of sets of elements in the semantics in addition to sets of change operations.

With these basic operations, an algorithm for computing exhibited witnesses following Eq. (2) can be written in one line: `return` $(sem(m_2) \setminus sem(m_1)) \cap sem(m_1 \oplus C)$. Naive algorithms for computing witnesses the set $C \in \Delta(m_1, m_2)$ is necessary or sufficient for, can directly operationalize their respective characterizations. Algorithm 4 computes all witnesses that have C as a necessary set of change operations. The algorithm follows the characterization in Eq. (1). It encodes the removal of the union of semantics of different models without the application of C (right part of Eq. 1) as an iteration over these models and removal of the elements in their semantics (l. 6).

An analogous algorithm to compute the set of witnesses that a given set of change operations is sufficient for similarly encodes Eq. (3). The required changes in Algorithm 4 are checking for containment in line 3 and replacing subtraction with intersection in line 6.

6 Application to languages and differencing operators

We now show how Diffuse, the language-independent, abstract framework defined above, can be applied to concrete modeling languages. We demonstrate instantiations of the Diffuse framework for ADs, CDs, and FMs. These modeling languages are all very different in their concrete syntax and semantics.

The following subsections present several examples.

6.1 Relating activity diagrams syntactic and semantic changes

We instantiate the relations and operators defined in Sect. 5 for UML ADs.

6.1.1 AD language

ADs define activities, the actions they are composed of, and their data flow and control flow. They are widely used for behavior modeling, and in variants (BPMN), they provide means for the description of processes and workflows. We denote the syntactic domain of ADs by *AD*.

We define the semantics of an AD as the set of execution traces it allows for a given data input provided by the environment. The execution traces in the semantic domain of ADs consist of the names of actions executed in the activity. We denote the domain of execution traces by T and the semantic mapping that associates an AD with the execution traces in its semantics as the function $sem_{AD} : AD \to \wp(T)$. An operational semantics of ADs based on a translation to SMV following the above semantics definition is available from [27] and has been applied in an implementation of a semantic differencing operator for ADs [26].

As an approach for syntactic differencing of ADs, we adapt a framework based on *compound operations* for BPMN models introduced by Küster et al. [21]. Examples of the change operations[2] presented in [21] are `insertAction(`*name*`,` *pred*`,` *succ*`)` to insert the action node labeled *name* between node *pred* and *succ*, `moveAction(`*name*`,` *pred*`)` to move the action named *name* after node *pred*, and `insertCyclicFragment(`*pred*`,`*frag*`)` to insert the cyclic fragment *frag* after node *pred*. These operations are called compound operations because they perform additional necessary steps to obtain well-formed models. As an example, the operation `moveAction` removes an action, adds a control flow edge between its former predecessor and successor, and also inserts new control flow edges at the destination position of the action.

[2] Removed first parameter (process model) and successor node parameter (in addition to *dest*) when clear in examples.

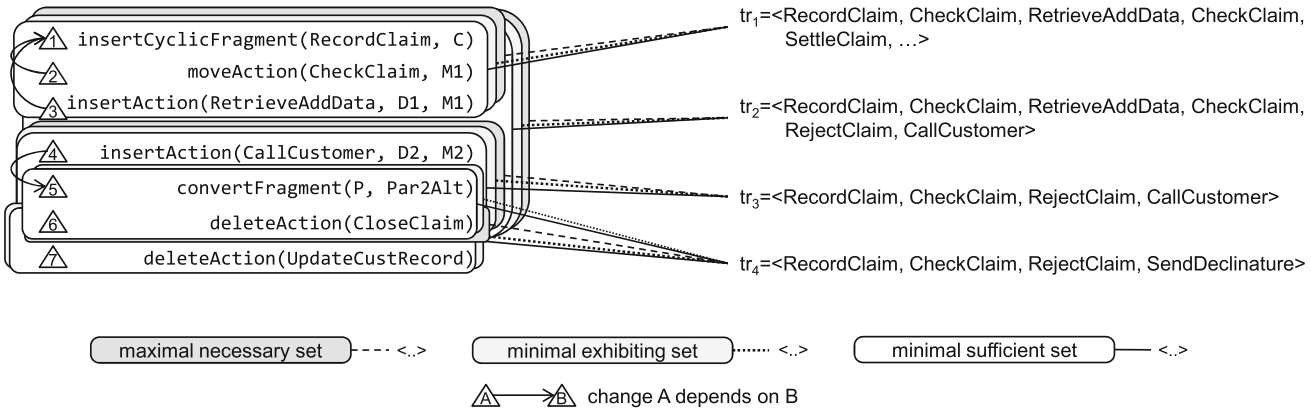

Fig. 6 Change operations from ad_1 to ad_2 as shown in Fig. 1, with application dependencies, semantic diff witnesses, and identification of minimal sufficient, minimal exhibiting, and maximal necessary sets of change operations. Not all witnesses are shown

6.1.2 AD application example

We illustrate the relation between the syntactic changes from ad_1 to ad_2 as shown in Fig. 1 and resulting semantic diff witnesses in Fig. 6. The left side of Fig. 6 shows a complete set of change operations and their application dependencies, e.g., the change operation `moveAction(CheckClaim, M)` depends on the insertion of the cyclic fragment `C` (here consisting of nodes `M1` and `D1`). The right side shows diff witnesses in the semantics of ad_2 but not ad_1.

Solid lines in Fig. 6 connect minimal sufficient sets of change operations (in white boxes) and traces added to the semantics of the second model through their application. Pointed lines connect minimal exhibiting sets of change operations (in light gray boxes) and their witnesses. Dashed lines connect maximal necessary sets of change operations (in gray boxes) and their witnesses. In this example for diff witnesses tr_1, tr_2, and tr_3 the individual minimal sufficient, minimal exhibiting, and maximal necessary sets coincide, i.e., applying the necessary change operations already exhibits and guarantees the existence of the witnesses regardless of other change operations applied. An interesting case is demonstrated by diff witness tr_4. The diff witness has two different but overlapping minimal sufficient sets of change operations. The maximal necessary set of change operations for tr_4 consists of the single element in the intersection of both minimal sufficient sets. The maximal necessary set of changes alone is not exhibiting the witness tr_4. Its minimal exhibiting sets are its two minimal sufficient sets.

6.2 Relating class diagrams syntactic and semantic changes

We instantiate the relations and operators defined in Sect. 5 for UML CDs.

6.2.1 CD language

CDs define classes, their properties, and the relations between them. They are widely used in object-oriented design, and its variants Ecore and MOF are prevalent and standardized means for the definition of metamodels of modeling languages. We denote the syntactic domain of CDs by *CD*.

The semantics of a CD is the set of object models (OM) it allows to instantiate from its classes. Object models contain objects with concrete values for attributes defined in the CD and links between objects that conform to associations of the CD in direction and multiplicities. We denote the semantic domain OM of CDs by *OM* and the semantic mapping from a CD to the OMs it allows to instantiate as the function $sem_{CD} : CD \rightarrow \wp(OM)$. A definition of CD semantics based on a translation to Alloy is available, e.g., from [28]. Implementations of the semantic differencing operator for CDs were presented in [22,29].

As an approach for syntactic differencing of CDs, we choose a framework based on low-level edit operations as defined by Alanen and Porres [2] with a CD-specific formalization [33]. Rindt et al. [33] formalize edit operations on CDs as graph transformations implemented with EMF-Henshin. The list comprises around 100 operations, e.g., `createClass(`*cName*`)` to create a new class with name *cName*, `createGeneralization(`*parent*, *child*`)` to declare *child* a subclass of *parent*, and `moveAssociationEnd(`*eName*, *target*`)` to move association end *eName* to class *target*.[3]

6.2.2 CD application example

We illustrate the relation between the syntactic changes from cd_1 to cd_2 as shown in Fig. 2 and resulting semantic diff witnesses om_1 to om_4 in Fig. 7. The left side

[3] Shortened names of CD change operations defined in [33]

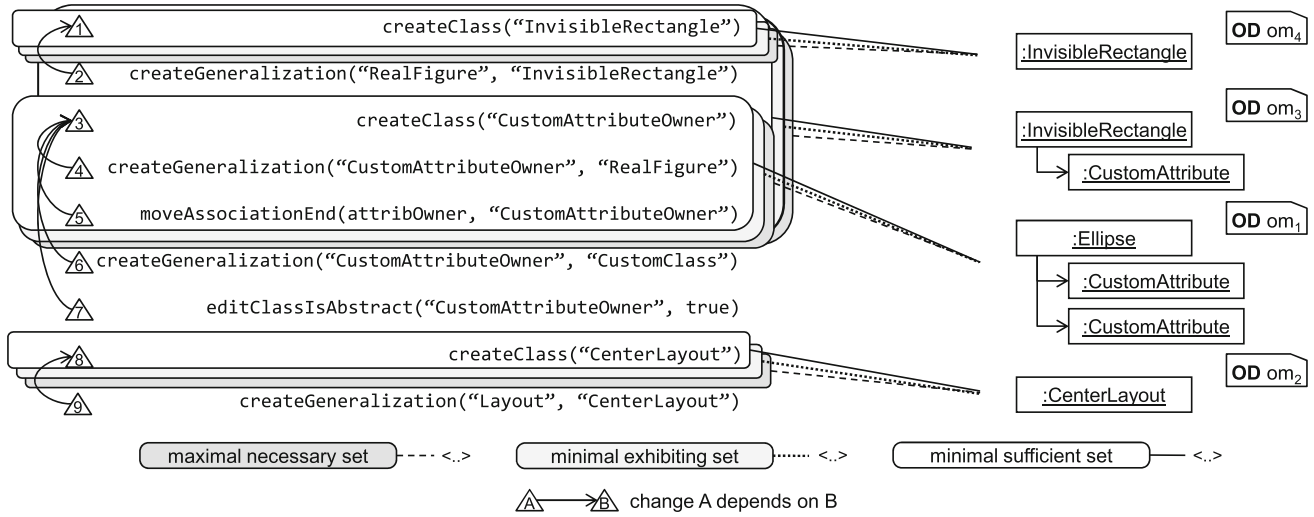

Fig. 7 Change operations from cd_1 to cd_2 as shown in Fig. 2, with application dependencies, semantic diff witnesses from Fig. 3, and identification of minimal sufficient, minimal exhibiting, and maximal necessary sets of change operations. Not all witnesses are shown (there are infinitely many witnesses)

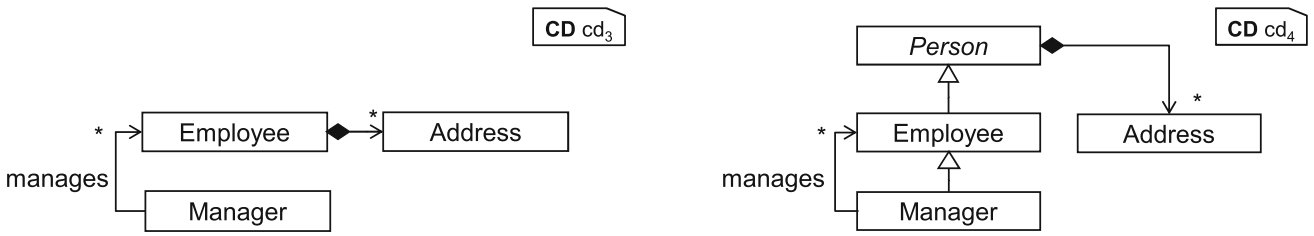

Fig. 8 Two CDs cd_3 and cd_4 modeling employees and management adapted from similar examples of [29]

of Fig. 7 shows a complete set of change operations and their application dependencies, e.g., the change operation `createGeneralization("RealFigure", "InvisibleRectangle")` depends on the creation of the class `InvisibleRectangle`. The right side lists diff witnesses in the semantics of cd_2 but not cd_1.

Solid lines in Fig. 7 connect minimal sufficient sets of change operations and their OM diff witnesses. Pointed lines connect minimal exhibiting sets of change operations (in light gray boxes) and their witnesses. Dashed lines connect maximal necessary sets of changes and their diff witnesses. Again, for the witnesses om_4, om_3, om_1, and om_2 also shown in Fig. 3 and used as examples throughout this paper, the minimal sufficient and the maximal necessary sets for all shown witnesses coincide. Following Corollary 1, these sets are also unique minimal exhibiting sets of change operations. The last change operation `createGeneralization("Layout", "CenterLayout")` shown in Fig. 7 is not part of any minimal sufficient, minimal exhibiting, or maximal necessary set for any witness in the semantic difference since the generalization has no effect on instances of cd_2 as shown in Fig. 2.

As an additional example for the relation between syntactic changes and semantic effects, consider the CDs shown in Fig. 8 adapted from similar examples of [29]. The CDs show a simple model of employees with addresses and managers who manage employees.

The data format of the ERP system of a company has been adapted from cd_3 to cd_4: The ownership of `Address` by `Employee` has been pulled up to an abstract class `Person` added in cd_4, and managers are now also employees (`Employee` identified as generalization of `Manager` in cd_4).

While performing some tests with cd_4, an engineer discovers that now managers may manage themselves while they were previously not managed. A witness of this change in the semantics is shown as om_5 in Fig. 9. The engineer wants to know which syntactic changes from cd_3 to cd_4 listed in Fig. 9 relate to this difference. She finds out that a minimal sufficient set of change operations for diff witness om_5 is the singleton set containing `createGeneralization("Manager", "Employee")`.

The engineer is unsure whether the change operation should be reverted. To further understand its effect on the

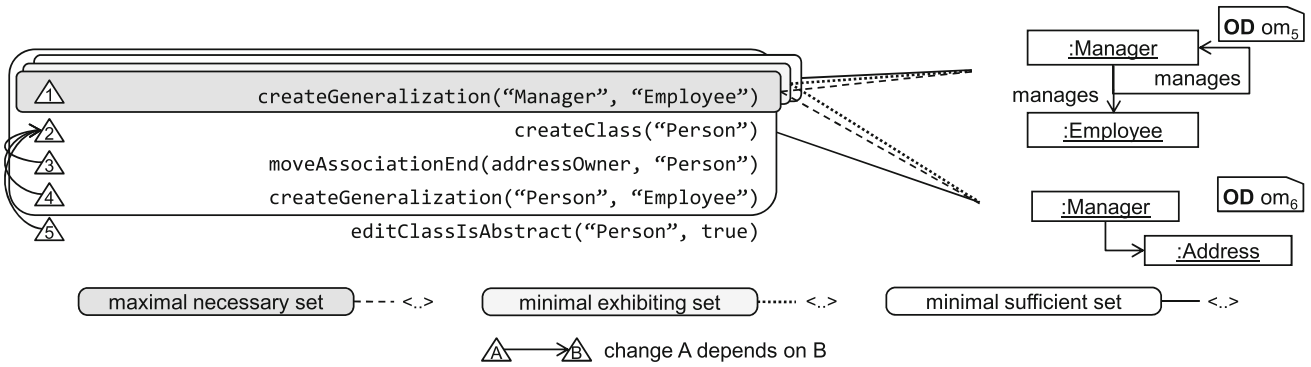

Fig. 9 Change operations from cd_3 to cd_4 as shown in Fig. 8, with application dependencies, semantic diff witnesses $om_5, om_6 \in \delta(cd_3, cd_4)$, and identification of minimal sufficient, minimal exhibiting, and maximal necessary sets of change operations. Not all witnesses are shown (there are infinitely many witnesses)

semantics she checks whether there are any other diff witnesses the change operation is necessary for. An analysis reveals that the change operation is in the maximal necessary set of om_6 (Fig. 9), which shows an instance of `Manager` that owns an instance of `Address`. In a review meeting, the team decides based on this information to not revert the change operation.

We illustrate the relation between the syntactic changes from cd_3 to cd_4 as shown in Fig. 8 and resulting semantic diff witnesses in Fig. 9. The left side of Fig. 9 shows a complete set of change operations and their application dependencies, e.g., change operations numbered 3-5 depend on the change operation `createClass("Person")`. The right side shows diff witnesses in the semantics of cd_4 but not cd_3. The OMs of the witnesses are also depicted as ODs in Fig. 9.

Figure 9 illustrates the relations of necessary, exhibiting, and sufficient sets of change operations as above. An interesting case is diff witness om_6 where the maximal necessary set of changes is not empty but smaller than the minimal sufficient set of change operations. The exclusive application of change operation 1 does exhibit om_6 in the semantics of the new CD, i.e., the set {1} is maximal necessary and minimal exhibiting. The set is, however, not a sufficient set because the application of change operations {1, 2, 3} leads to a CD without om_6 in its semantics as previously illustrated for om_6 in Fig. 5.

6.3 Relating feature models syntactic and semantic changes

We instantiate the relations and operators defined in Sect. 5 for FMs.

6.3.1 FM language

FMs define features and their inclusion and exclusion dependencies in valid product configurations. They are a popular description mechanism in product line engineering. We denote the syntactic domain of FMs by FM.

The semantics of a FM is defined in terms of the configurations it allows. A configuration is a valid set of features that meets all inclusion and exclusion constraints of the FM. We denote the semantic domain of FMs by $Conf$ and the semantic mapping from a FM to the configurations it allows as the function $sem_{FM} : FM \rightarrow \wp(Conf)$. A definition of FM semantics based on translations to SAT and Binary Decision Diagrams (BDDs) is available from [8] and has been applied in a definition of a semantic differencing operator for FMs [1].

As an approach for syntactic differencing of FMs, we choose a framework based on elementary FM-specific edit operations, as used in the work of Thüm et al. [36]. Examples of change operations are `moveTo(`f`,` $parent$`)` to move feature f to parent feature $parent$, `makeOptional(`f`)` to make the inclusion of feature f on the selection of its parent optional, and `addConstraint(`c`)` and `removeConstraint(`c`)` to add and remove custom constraints between features.

6.3.2 FM application example

We illustrate the relation between the syntactic changes from fm_1 to fm_2 shown in Fig. 4 and resulting semantic diff witnesses in Fig. 10. The left side of Fig. 10 shows a complete set of change operations. The right side lists configurations from the semantic difference between fm_1 and fm_2. Not all diff witnesses are shown.

Solid lines in Fig. 10 connect minimal sufficient sets of change operations (in white boxes) and configurations added to the semantics of the second model through their application. Pointed lines connect minimal exhibiting sets of change operations (in light gray boxes) and their witnesses. Dashed lines connect maximal necessary sets of change operations (in gray boxes) and their witnesses.

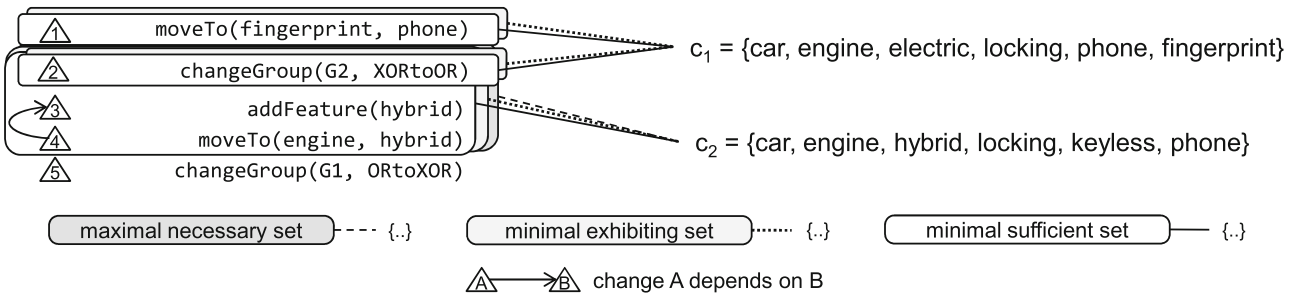

Fig. 10 Change operations from fm_1 to fm_2 from Fig. 4 with application dependencies, semantic diff witnesses, and identification of minimal sufficient, minimal exhibiting, and maximal necessary sets of change operations. Not all witnesses are shown

The configuration c_1 is an example for a diff witness with two disjoint sets of minimal sufficient change operations. The minimal sufficient sets of c_1 are also minimal exhibiting sets. Disjoint sufficient sets imply that the diff witness has an empty necessary set of change operations. For witness c_2, again the maximal necessary, minimal exhibiting, and minimal sufficient sets of change operations coincide. The operation `changeGroup(G1, ORtoXOR)` is not contained in any minimal sufficient, minimal exhibiting or any maximal necessary set of change operations for any witness in $\delta(fm_1, fm_2)$ because it is a refinement of the FM semantics and does not add diff witnesses.

7 Extensions

We present three extensions to the basic Diffuse framework, as defined in Sect. 5. First, we consider grouping change operations in the syntactic difference to obtain a higher-level representation. Second, we motivate to consider elements removed from the model's semantics instead of only those added and discuss the impact of this extension on the Diffuse framework. Finally, we discuss the explanation of change operations with witnesses on a more fine-grained level than that of the relations defined in Sect. 5.

7.1 Grouping of syntactic differences

One challenge for the comprehension of and interaction with differences is the potentially high number of change operations and related diff witnesses. In addition, our generic algorithms presented in Sect. 5.2 for computing the relations have a time complexity exponential in the size of the syntactic difference measured as number of change operations.

To support comprehension and possibly lower computation times by considering less combinations of change operations, we suggest to group related syntactic change operations. Relatedness can be defined in different ways. We give three examples and discuss the impact of grouping on the three relations defined in the Diffuse framework.

7.1.1 Grouping examples

As an example, consider the set of change operations consisting of creating a new class `P`, creating a generalization relation with an existing class `C`, and making the parent class `P` abstract. This set of operations can be grouped to a user-level edit operation `createAbstractParent(P,C)` following a framework for "semantic lifting" of change operations introduced by Kehrer et al. [18]. This generic framework is directly applicable in our setting, and the above single lifting rule reduces the number of change operations in the syntactic difference from cd_3 to cd_4 shown in Fig. 9 from six to three change operations by introducing `createAbstractParent("Person", "Employee")`. Another instance of this pattern, `createAbstractParent("CustomAttributeOwner", "RealFigure")`, appears in the syntactic difference from cd_1 to cd_2 shown in Fig. 7.

An alternative is to group together dependent change operations. If a change operation depends on multiple independent change operations, these are grouped into the same group to ensure partitioning of the set of change operations. As an example, the change operations from ad_1 to ad_2 listed in Fig. 6 could be grouped as {1, 2, 3}, {4, 5}, {6}, and {7} reducing the number of change operations from seven to four.

Finally, and as a third example, syntactic low-level edit operations can be grouped to compound edit operations that express language-specific atomic changes. One example is the framework for compound change operations of process models introduced by Küster et al. [21]. We have already used this framework in our AD examples. As described in [21], the operation `insertAction(`*name*`,` *pred*`,` *succ*`)` is composed of four operations: deleting the edge between nodes *pred* and *succ*, inserting node *name*, adding an edge from node *pred* to node *name*, and adding an edge from node *name* to node *succ*.

The choice of a suitable grouping criteria for change operations depends on the analysis of interest and modeling language considered. In the case of ADs, the grouping was already a part of the syntactic differencing framework.

7.1.2 The impact of grouping on the diffuse framework

We now discuss the impact of grouping of change operations on the relations defined in Sect. 5. A grouping method `group(.)` that partitions a set of change operations $\Delta(m_1, m_2)$ is a unique mapping of $C' \subseteq \Delta(m_1, m_2)$ to a set of groups of change operations such that $\forall s \in \texttt{group}(C') : s \cap C' \neq \emptyset$. The mapping `group(.)` is unique because every change operation appears in exactly one partition of $\Delta(m_1, m_2)$. Note that the set $\texttt{ungroup}(\texttt{group}(C')) := \bigcup_{s \in \texttt{group}(C')} s$ contains and in general might be larger than C'.

In some cases, grouping leads to loss of information compared to the individual change operations. As an example, consider the changes from cd_3 to cd_4 and the witness om_6 all shown in Fig. 9. The first four out of five change operations are minimal sufficient for the witness om_6. When grouping dependent changes, the minimal sufficient set for om_6 contains all change operations. Applying grouping by lifting change operations using `createAbstractParent(P,C)` has the same effect for om_6.

We have suggested grouping as a means to reduce the number of change operations and speed up computation. It is thus interesting to investigate the validity of computation results when going back to individual change operations via the operator `ungroup(.)`. Ungrouped exhibiting sets of change operations are exhibiting sets of change operations because after ungrouping the witness still appears in the semantics of the model obtained from applying all change operations.

There is, however, no guarantee that ungrouped sufficient sets are sufficient sets of change operations for a witness. As an example, consider the changes from cd_3 to cd_4 shown in Fig. 9, the witness om_6, and grouping by dependent changes into partitions $\{1\}$ and $\{2, 3, 4, 5\}$. The set of groups $\{\{1\}\}$ is sufficient for om_6, while $\texttt{ungroup}(\{\{1\}\}) = \{1\}$ is not. In the ungrouped case, the application of $\{1, 2, 3\}$ to cd_3 does not exhibit om_6 and thus $\{1\}$ is not a sufficient set of change operations for om_6.

Similarly, there is no guarantee that ungrouped necessary sets are necessary sets of change operations for a witness. As an example, consider the changes from cd_1 to cd_2 and the witness om_2 all shown in Fig. 7 and grouping of dependent changes. A grouped necessary set of om_2 is $\{\{8, 9\}\}$. The set $\{8, 9\}$ is, however, larger than the maximal necessary set of change operations for om_2 ($\{8\}$ as shown in Fig. 7).

In the other direction when going from change operations to groups of change operations using the operator `group(.)`, the only preserved relation is that of necessary sets of change operations. This follows directly from Definition 1 and that grouping partitions change operations. When grouping an exhibiting set of change operations, the result is not necessarily exhibiting, e.g., grouping may add change operations such that the application will not lead to a well-formed model as required by Definition 2. For the same reason, grouping also does not necessarily preserve the relation of sufficient change operations.

It might be possible to define grouping criteria for change operations that preserve additional relations when grouping and ungrouping. It is also possible that results of computing the grouped relations can be reused in refinement steps that remove grouping. We leave these topics to future work.

7.2 Elements removed from semantics

The framework we have presented so far builds on the definition of semantic difference as the elements that were added to the semantics of the second model. Many changes, however, also remove elements from the semantics of a model. As examples, consider a change operation in a CD that makes a class abstract, as shown in Fig. 9, or a change operation in a FM that changes an alternative to an exclusive alternative, as shown in Fig. 10. These change operations are refinements in the context of the models they are applied to, i.e., they only remove elements from the semantics of the models.

The set of elements removed from the semantics following a change from model m_1 to model m_2 is the set $sem(m_1) \setminus sem(m_2)$, i.e., the elements returned by the application of the semantic differencing operator with reversed arguments $\delta(m_2, m_1)$. We present an extension of the framework by extending the relation of sets of change operations to positive witnesses with negative witnesses, i.e., elements that were in the semantics of the first model and are not in the semantics of the second. The corresponding relations from Definition 1-3 are presented in Definition 4.

Definition 4 *(Relations for removed elements)* For models $m_1, m_2 \in \mathcal{M}$, and a removed element $r \in sem(m_1)$ with $r \notin sem(m_1)$ a set $C \subseteq \Delta(m_1, m_2)$ is:

- a **necessary** set of change operations iff $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : r \notin sem(m_1 \oplus C') \Rightarrow C \subseteq C'$,
- an **exhibiting** set of change operations iff $(m_1 \oplus C) \in \mathcal{M}$ and $r \notin sem(m_1 \oplus C)$, and
- a **sufficient** set of change operations iff $(m_1 \oplus C) \in \mathcal{M}$ and $r \notin sem(m_1 \oplus C)$ and $\forall C' \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C') \in \mathcal{M} : C \subseteq C' \Rightarrow r \notin sem(m_1 \oplus C')$.

The properties for necessary, exhibiting, and sufficient sets of change operations in Definition 4 follow the same pattern as their counterparts for diff witnesses. A set of change operations is necessary for the removal of r iff it is contained in all sets of change operations whose applications lead to removal of r from the semantics of the model. A set of change operations exhibits the removal of r if its application removes

r from the semantics of the model. Finally, a set of change operations is sufficient for the removal r iff its inclusion in the set of applied change operations guarantees that r will not appear in the semantics of the model after applying more change operations from $\Delta(m_1, m_2)$.

As an example, consider the trace $tr_5 =$ <RecordClaim, CheckClaim, RejectClaim, SendDeclinature, UpdateCustRecord, CloseClaim> possible in ad_1 but not in ad_2 shown in Fig. 1. The maximal necessary set of change operations for the removal of this trace is the deletion of the action node labeled CloseClaim. A minimal sufficient set is the deletion of CloseClaim and the conversion of fragment P from parallel to alternative.

As another example, consider the change from fm_1 to fm_2 shown in Fig. 4. The configuration {car, engine, electric, gas, locking, fingerprint} is possible in fm_1 but not in fm_2. The set of change operations that consists of moving the feature fingerprint and changing group G1 to an exclusive alternative is a maximal necessary, minimal sufficient, and minimal exhibiting set of change operations for this removed element.

Finally, the changes from cd_1 to cd_2 shown in Fig. 2 and the changes from cd_3 to cd_4 shown in Fig. 8 did not remove any elements from the semantics of the first CD, and thus, no removed element exists to relate to sets of change operations.

The relations defined in Definition 4 for removed elements are very similar to the ones of diff witnesses defined in Defs. 1-3. The major difference is the negation of the containment of the element $r \in sem(m_1)$ in the semantics of the model after the application of change operations.

Interestingly, all of the properties formalized in theorems in Sect. 5 hold when replacing the witness $w \in \delta(m_1, m_2)$ with the removed element $r \in \delta(m_2, m_1)$. Subsets of the necessary sets of change operations for a removed element are also necessary sets of change operations (compare to Theorem 1), and the maximal necessary set of change operations for every removed element is unique (compare to Theorem 2). The containment of necessary in exhibiting and sufficient sets of change operations holds also for removed elements (compare to Theorem 3 and Theorem 5). Finally, extensions of sufficient sets of change operations whose application leads to well-formed models are sufficient sets for a removed element (compare to Theorem 4).

The duality of diff witnesses and removed elements allows the reuse of the algorithms presented in Sect. 5.2 with minor changes. Algorithm 1 computes the maximal necessary set of change operations for a removed element when negating the membership check in l. 5. Algorithm 2 computes all exhibiting sets of change operations for a removed element when negating the membership check in l. 5. Algorithm 3 computes the minimal sufficient sets of change operations for a removed element when negating the membership checks in l. 6 and l. 10.

Similarly, the characterization of diff witnesses from Sect. 5.3 can be adapted to characterize elements removed from the semantics of a model.

The removed elements a set $C \subseteq \Delta(m_1, m_2)$ is necessary for are:

$$\delta(m_2, m_1) \cap \left(\bigcap_{\substack{C \nsubseteq C' \subseteq \Delta(m_1, m_2) \\ m = m_1 \oplus C' \in \mathcal{M}}} sem(m) \right).$$

The formalization of removed elements starts from the set of removed elements (semantic differencing operator in reversed application order) and retains all elements in the semantics of all models after applying sets of change operations not containing all changes in C. A removed element remains in the set if none of the other sets of change operations remove the element from the semantics of the model.

The removed elements a set $C \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C) \in \mathcal{M}$ is exhibiting are:

$$\delta(m_2, m_1) \setminus sem(m_1 \oplus C).$$

This characterization starts from the set of removed elements from model m_1 to model m_2 and removes all elements in the semantics of the model after application of C. This leaves removed elements also removed in $m_1 \oplus C$.

The removed elements a set $C \subseteq \Delta(m_1, m_2)$ with $(m_1 \oplus C) \in \mathcal{M}$ is sufficient for are:

$$\delta(m_2, m_1) \setminus \left(\bigcup_{\substack{C \subseteq C' \subseteq \Delta(m_1, m_2) \\ m = m_1 \oplus C' \in \mathcal{M}}} sem(m) \right).$$

The characterization starts from the set of removed elements and removes all that are in the semantics of any other model obtained when applying at least the change operations in C.

We believe that this extension of the Diffuse framework with removed elements provides valuable information for understanding the impact of change operations. Note that the semantic differencing operator $\delta(m_1, m_2)$ provides this information by applying it with reversed arguments $\delta(m_2, m_1)$. The extension we presented in this subsection is, however, not simply reversing the arguments of the differencing operators of the framework. Syntactic differencing operators are usually not symmetric, indeed, none of the ones applied in Sect. 6 is. Thus, relating elements removed from the semantics to change operations requires extensions as presented in this subsection.

7.3 Explanation of change operations with witnesses

When trying to understand a set of change operations, the relations defined in Sect. 5 provide sets of related diff witnesses. Based on the task at hand, not all relations are equally informative. As an example, when considering to revert some change operations, a set of witnesses these operations are necessary for are of interest for a decision. As another example, when one wants to apply only a subset of change operations, the witnesses these are exhibiting are helpful in choosing the subset.

The number of witnesses exhibited by a set of change operations might be very large, and many witnesses may also appear when looking at smaller sets of change operations. Consider the CDs cd_1 and cd_2 shown in Fig. 2 and the application of the first five change operations shown in Fig. 7: creating classes `InvisibleRectangle` and `CustomAttributeOwner` and moving the association end `attribOwner` to class `CustomAttributeOwner`. This set of change operations exhibits the witness om_4, an instance of class `InvisibleRectangle`, shown in Fig. 3. The same witness is also exhibited when only applying the first out of five change operations, i.e., creating the class `InvisibleRectangle`. Another exhibited witness, which is more informative for the set of all five change operations, is om_3 shown in Fig. 3. This witness is not exhibited by any smaller set of change operations, and the considered change operations are a necessary set of change operations for om_3.

7.3.1 Most informative exhibited witnesses

When explaining sets of change operations, we are interested in the most informative exhibited witnesses. In this context, *most informative* relates witnesses and sets of change operations.

Following the example above, we suggest to present witnesses that a set of change operations $C \subseteq \Delta(m_1, m_2)$ exhibits but not those exhibited by smaller sets $C' \subsetneq C$ of change operations, i.e., witnesses that have C as a minimal exhibiting set. In the example of cd_1, cd_2, the first five change operations shown in Fig. 7 are an exhibiting set for om_1, om_3, and om_4, but only for om_3 they are a minimal exhibiting set (see relations illustrated in Fig. 7). Thus, we consider om_3 as the most informative exhibited witness to explain these change operations.

We formally define the notion of most informative witness in Definition 5.

Definition 5 *(Most informative exhibited witnesses)* For two models $m_1, m_2 \in \mathcal{M}$, the most informative exhibited witnesses of $C \subseteq \Delta(m_1, m_2)$ are witnesses $W \subseteq \delta(m_1, m_2)$ with minimal exhibiting set C.

Does every set of change operations $C \subseteq \Delta(m_1, m_2)$ have most informative exhibiting witnesses? By definition, all diff witnesses have at least one minimal exhibiting set of change operations. However, some sets of change operations are not minimal exhibiting sets for any witness. We discuss the possible cases below and suggest alternative informative witnesses for these sets.

7.3.2 Alternative informative witnesses

Given two models $m_1, m_2 \in \mathcal{M}$ and a set of change operations $C \subseteq \Delta(m_1, m_2)$, we can distinguish four cases:

1. C has most informative exhibited witnesses;
2. the application of C does not yield a well-formed model, i.e., $m_1 \oplus C \notin \mathcal{M}$;
3. $m_1 \oplus C \in \mathcal{M}$ but C does not exhibit any witnesses, i.e., $sem(m_1 \oplus C) \cap \delta(m_1, m_2) = \emptyset$; or
4. C is an exhibiting, but not minimal exhibiting set for any witness.

The first case is the case discussed above and illustrated in the example. Most informative witnesses exist and are those defined in Definition 5.

The second case can be handled by computing minimal extensions of C that lead to well-formed models and present their informative witnesses according to cases 1, 2, or 4. Consider the example of ad_1 and ad_2 and their syntactic change operations shown in Fig. 6. The application $ad_1 \oplus \{3\}$, i.e., inserting the action `RetrieveAddData` after node `D1`, is not defined (case 2) because node `D1` does not exist in ad_1. The only minimal extension C' of $C = \{3\}$ with $ad_1 \oplus C' \in \mathcal{M}$ is $C' = \{1, 3\}$ with a most informative witness `<RecordClaim, CheckClaim, RetrieveAddData, SettleClaim, ... >` (case 1).

The third case means that the application $m_1 \oplus C$ either is a refinement of m_1 or that $\delta(m_1, m_1 \oplus C)$ yields elements in the semantics of $m_1 \oplus C$ that are not diff witnesses in the semantic difference of m_1 and m_2. In this case, no witness can be shown to explain the set of change operations C.

Finally, the fourth case means that some change operations in C do not contribute to exhibit additional witnesses. The set C, however, does exhibit witnesses. In this case, informative witnesses can be defined as informative witnesses of the maximal strict subsets of C according to cases 1 to 4.

7.3.3 Beyond explanation by exhibited witnesses

One might also consider explanations based on the two other relations defined in Sect. 5. An explanation based on necessity of change operations C potentially includes witnesses exhibited for extensions of C (see Definition 1). An explanation based on sufficiency of change operations C potentially

yields less witnesses compared to the explanation based on the exhibiting relation because Definition 3 is stronger than Definition 2.

The explanation of change operations by witnesses can be combined easily with an explanation by elements removed from the semantics. We have presented this dual relation in Sect. 7.2, Definition 4. Specifically, in the third case above when $m_1 \oplus C$ refines m_1 and no exhibited witness can be used to explain the change operations, removed elements might contribute an explanation.

Finally, the explanation of change operations can be generalized from diff witnesses. In case of manipulation of change operations, e.g., in the context of merging, a user might be interested in effects on the semantics regardless of the semantics of a second model. This generalization requires a generalization of Definition 2 of exhibiting sets of change operations from diff witnesses to all elements in the semantic domain.

8 Discussion

We discuss challenges and limitations of the Diffuse framework.

8.1 Modeling language semantics

The Diffuse framework applies to modeling languages with set-based semantics, i.e., the language definition includes a semantic mapping that defines the semantics of a well-formed model as a set of elements in its semantic domain (see Sect. 4).

Many languages have semantics definitions of this form, e.g., the configurations possible to instantiate from a FM. Most languages allow different ways to define their semantics, e.g., the semantics of ADs can be defined using symbolic transition systems [10] or using sets of possible execution traces. To apply Diffuse, only the latter definition makes sense. Finally, there are some languages for which set-based semantics are less natural or have not been defined, e.g., markup languages. The Diffuse framework does not currently support such languages.

Many existing implementations of semantic differencing operators [1,22,26,29] (including ours) establish a relation between names used in models and names in the semantic domain. As a result, the change of a name in the model can produce many diff witnesses. One might consider this an undesirable result. However, it is important to note that the general approach of a semantics definition by a semantic mapping, and thus the approach chosen in the Diffuse framework, does not prescribe such a tight relation between names. Finally, the Diffuse framework, unlike other approaches, helps to reveal the relation between the syntactic rename and the resulting diff witnesses.

8.2 Syntax and change operations

The Diffuse framework applies to modeling languages with syntactic change operations that are partially ordered. We denote the application of a set of changes by the operator $\oplus$. This operator takes the order of change operations into account. All definitions in this paper require that the application of $\oplus$ is well defined, i.e., there is either no or exactly one resulting model.

Most syntactic differencing frameworks [2,19,21,35] have this property. Note that the order of change operation application might depend on the model and set of change operations. In some cases, it even depends on the current application state, i.e., an intermediate model after application of change operations [5,20,24]. In the examples shown in Sect. 6, the orders illustrated in Figs. 6, 7, 9 and 10 depend only on the original model and the change operations.

An extension of the Diffuse framework might be able to support a larger set of syntactic change operation frameworks by allowing multiple resulting models from the same change operation application. This extension appears only relevant when considering merge scenarios where the final model is not uniquely determined. In the differencing cases we consider in this paper, the application of change operations $\Delta(m_1, m_2)$ to model m_1 in any allowed order leads to the second model m_2.

8.3 Restriction to differences

For two models $m_1, m_2 \in \mathcal{M}$, the syntactic difference $\Delta(m_1, m_2)$ in terms of syntactic change operations is not necessarily unique. However, most existing syntactic differencing approaches compute and present only one set of change operations [2,7,20,21,35]. The Diffuse framework relates syntactic differences $\Delta(m_1, m_2)$ and semantic differences $\delta(m_1, m_2)$ of models m_1 and m_2. Thus, all the definitions and algorithms we presented in Sect. 5 and Sect. 7 apply to the model differences $\Delta(m_1, m_2)$ and $\delta(m_1, m_2)$, and not to arbitrary sets of change operations or to elements from the semantics of intermediate models.

Without the restriction to differences $\Delta(m_1, m_2)$ and $\delta(m_1, m_2)$, the relations of necessary, exhibiting, and sufficient sets of syntactic change operations for elements of the semantic domain could still be relevant but would serve a different purpose. As an example, an exhibiting set of change operations for an arbitrary element of the semantics domain, would represent a possible syntactic update that will modify the semantics of a model to include this element.

8.4 Properties and special cases

Diffuse is a generic language-independent framework for relating syntactic and semantic differences. We have dis-

cussed properties that hold for all instances of the framework and some that do not hold in general. This leaves opportunities for analyzing special cases. Specifically, the algorithms and characterizations presented in Sect. 5.2 and Sect. 5.3 apply to all instances of the framework but might be replaced by simpler or more efficient versions depending on the choice of a language and its syntactic and semantic differencing operators.

Some interesting questions are:

- When do algorithms with a better time complexity exist?
- When do the maximal necessary and minimal sufficient sets of change operations coincide?
- When does the same change operation appear in all exhibiting sets of all witnesses?

Answers to these questions might contain combinations and restrictions of all parts of the Diffuse framework including the syntactic and semantic differencing operators.

8.5 Alternative necessary change operations

We have motivated a use case for the set of necessary change operations to omit change operations and prevent a witness from appearing in the semantics of a model. We have also presented examples for empty sets of necessary change operations. The sets of necessary change operations for a witness may be empty because different sets of change operations independently exhibit the witness.

The suggested use case of identifying change operations to omit the witness can thus not be completely handled by the presented relation of necessary change operations. The relation might be extended for this task to one that considers sets of sets of necessary change operations. In the example of FMs fm_1, fm_2, and the witness c_1 shown in Fig. 10 a maximal necessary set of sets would be {{1}, {2}} (change operations as numbered in Fig. 10). To prevent the witness c_1 from appearing in the semantics of $fm_1 \oplus C$ with $C \subseteq \Delta(fm_1, fm_2)$, at least one change operation from every set has to be omitted from C.

8.6 Implementation and computation

Existing implementations of semantic differencing operators [1,22,26,29] exhibit high computational complexity for computing diff witnesses. The generic algorithms we present in Sect. 5.2 do not require the computations of diff witnesses but only the check whether a given witness is in the semantics of a model after the application of different change operations. However, both algorithms have a time complexity exponential in the size of the syntactic difference (typically at least an order of magnitude smaller than the models themselves).

We expect that both the iteration of subsets of the syntactic difference and the membership check of a witness in the semantics of a model allow for language-specific heuristics and optimizations. As an example, the iteration of change operation subsets that lead to well-formed ADs can be supported by known change operation application dependencies [14]. As another example, a sound, incomplete, but fast heuristics for membership of an OM in the semantics of a CD is the check whether all classes with instances in the OM exist in the CD.

8.7 Presentation and diff interaction

Our ultimate goal in relating syntactic and semantic differences is to assist engineers with information required for comprehension of model evolution and maintenance tasks. A tight integration into IDEs and model management tools will require new ways of interaction. Existing user interfaces, e.g., the ones in our prototypes for CD and AD semantic differencing [26,29], need to be extended with the computed relations and seamless presentation of both change operations and diff witnesses. We consider such an extension to be challenging, due to the different domains and several many-to-many relations involved.

In other previous work, we have generated natural language explanations to help engineers understand the reasons for inconsistencies [31]. Störrle [34] used natural language text to present CD differences. The relations between syntactic operations and semantic witnesses in the Diffuse framework suggest that a similar approach, to generate natural language supporting descriptions, may be effective here, too.

9 Related work

We summarize related work on syntactic differencing, on semantic differencing as discussed here, on alternative approaches to semantic differencing, and on works that combine syntactic and semantic differencing.

9.1 Syntactic differencing

Syntactic differencing operators and model comparison frameworks have been suggested by many authors, including Alanen and Porres [2], Kehrer et al. [19], Küster et al. [21], Lin et al. [23], and Taentzer et al. [35].

Alanen and Porres [2] introduce generic syntactic differencing algorithms and operators for OMG MOF-based models. Change operations are classified as creations, deletions, and modifications. Alanen and Porres also introduce a special union operator to ensure that change operations applied in different orders lead to the same model.

Lin et al. [23] use similar difference representations and introduce differencing algorithms for domain-specific languages with graph-based meta-models. The result of differencing includes a similarity matching of elements. Extensions of the Diffuse framework with matching requires a consistent integration with semantic differencing.

Taentzer et al. [35] present a graph-based differencing framework based on deletions and insertions. We have applied an instance of this framework for CDs, as defined by Rindt et. al [33], for the example in Sect. 6.2.

Kehrer et al. [19] introduce consistency-preserving edit scripts that are extracted from low-level graph modifications. The application of change operations ensures that the resulting intermediate model is well-formed, i.e., $m \oplus C \in \mathcal{M}$.

Küster et al. [21] define higher-level change operations specific to BPMN models and present an algorithm to compute them.

These frameworks range from language-independent low-level change operations [2,23,35] to language-specific higher-level and composed change operations [18,19,21,33], including CDs and ADs. Syntactic differencing has also been implemented in several tools, including, e.g., EMF Compare [9].

These works do not consider the semantic effects of the syntactic changes.

However, many of these works are suited to provide a foundation for the syntactic change operations part in the Diffuse framework. Indeed in Sect. 6, we used change operations of ADs as defined by Küster et al. [21], of CDs as defined by Rindt et al. [33], and of FMs as used by Thüm et al. [36].

9.2 Semantic differencing

Semantic differencing operators, e.g., for ADs, CDs, and FMs, which compute elements in the semantics of one model and not the other, have been defined and implemented in a number of works by us [26,29] and by others, Acher et al. [1] and Langer et al. [22].

Different techniques have been used for different languages. In [26], we have used a symbolic BDD-based algorithm to find the set of traces that is possible in one AD and is not possible in another. In [29] we have used a translation to SAT, via Alloy [17], to find object models that are in the semantics of one CD and are not in the semantics of another. Acher et al. [1] have used a translation to SAT to find product configurations that are in the semantics of one FM and are not in the semantics of another. Langer et al. [22] have presented a framework for semantic differencing based on traces obtained from executing behavioral semantics specifications. They instantiate their framework to compute semantic differences of ADs, CDs, and Petri nets.

Despite their differences, these works share several common challenges, including the need to efficiently compute and effectively present a high number (or even an infinite number) of witnesses. These important challenges have so far been only partly addressed (one example is [30]).

Moreover, all these works can be used not only to find diff witnesses (object models, traces, product configurations), but also to check for higher-level semantic relations between models, including refinement, abstraction, and equivalence. For example, one CD refines another if the semantics of the first (the set of object models instances in its semantics) is a subset of the semantics of the second; two ADs are semantically equivalent if they have the same set of traces in their semantics. These high-level relations provide important insights on the evolution of a model.

None of these works, however, has related syntactic and semantic differences. The Diffuse framework we present in this paper relies on the definitions of semantic difference and diff witnesses provided in these works (as demonstrated in the three instantiation examples we discuss in Sect. 6).

9.3 Alternatives and combinations

Fahrenberg et al. [12] motivate a different approach to semantic differencing, where the difference between models is expressed as a model in the same language, e.g., representing the difference of two CDs by a single CD [11]. They present a compositional algebra of CDs with subtyping, conjunctive and disjunctive merge and difference, where the operators are described by means of manipulations of syntactic elements of the diagrams with proven sound semantic implications.

Major differences between our work and [11] should be noted. First, while their work is specific to CDs, ours is general and can be applied, as we show, to many and very different modeling languages. Second, while the work of Fahrenberg et al. defines specific syntactic manipulations, ours maps syntactic manipulations to their semantic consequences, manifested using witnesses. Their compact representation of difference might, however, be an interesting complement to a fine-grained analysis of change operations and diff witnesses.

Fisler et al. [13] presented impact analysis for modified access control policies in the Margrave policy analyzer. They compare two versions of an access control policy and define the change–impact as actions allowed after the change. The work can be seen as an instance of semantic differencing without an analysis of individual change operations. Fisler et al. support queries on the set of diff witnesses, which could be an interesting extension to existing operators and to the Diffuse framework.

Briand et al. [6] presented impact analysis for UML models based on syntactic changes. In their context, the impact of syntactic changes is a set of syntactical elements (including

implementation code) that might require updates following these changes. In contrast, in the setting of the Diffuse framework the impact of syntactic change operations are semantic diff witnesses.

Semantic differencing of programs has been studied by Jackson and Ladd [16] for procedural code by presenting as diff witnesses dependencies of variables added in the new version.

Apiwattanapong et al. [4] studied differencing for object-oriented programs. They use control flow graphs extended with semantics of object-oriented concepts. These works neither compute explicit syntactic changes nor relate syntax and semantics.

Thüm et al. [36] compare FM edits relating their semantics, e.g., as refactorings or generalization, using a SAT solver. However, they do not explicitly map between specific syntactic change operations and their semantic consequences.

Kehrer et al. [18] lift low-level edit operations (e.g., the ones presented in [33] for CDs) to more conceptual descriptions of model modifications. However, these descriptions are still syntactic. We believe that the Diffuse framework may take advantage of lifted operations for the presentation of syntactic changes as discussed in Sect. 7.1.

Finally, Gerth et al. [14] use a semantic comparison of process models to detect false merge conflicts computed by syntactic comparisons (operations from [21] as used in Sect. 6.1). A false conflict exists if one change operation prevents the other from being applied but both applications allow the same set of traces after application. We leave for future work the lifting the relation between change operations and diff witnesses on the one hand and merge scenarios on the other hand.

10 Conclusion

We presented Diffuse, a novel, language-independent framework, which relates syntactic and semantic model differences. The Diffuse framework builds on the notion of necessary, exhibiting, and sufficient sets of change operations for semantic diff witnesses. We formalized the framework, proved several of its important properties, and provided examples that demonstrated its application to three different concrete modeling languages.

The introduction of the Diffuse framework opens the way for interesting future work directions. First, the operators defined in Sect. 5 can be implemented to extend previously presented prototype tools for semantic differencing, e.g., our own CDDiff and ADDiff tools [26,29]. Some of these operators may be difficult to compute efficiently, and so symbolic approaches or heuristics, as we have used in CDDiff and ADDiff, may be required.

Second, as we discuss in Sect. 8.7, the presentation of the relations we have defined between syntactic changes and diff witnesses to engineers within a modeling tool is challenging due to the different domains and several many-to-many relations involved. In addition to the two versions of a model, one needs to represent concrete change operations and diff witnesses. Thus, the representation has to combine and relate several languages, each with its own syntax (and semantics). We leave these for future work.

Acknowledgments We thank the anonymous reviewers of the MoDELS'15 conference paper and its present journal version for their helpful comments. Part of this work was done while SM was on sabbatical as visiting scientist at MIT CSAIL. JOR acknowledges support from a postdoctoral Minerva Fellowship, funded by the German Federal Ministry for Education and Research.

References

1. Acher, M., Heymans, P., Collet, P., Quinton, C., Lahire, P., Merle, P.: Feature model differences. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE). Lecture Notes in Computer Science, vol. **7328**, pp. 629–645. Springer (2012)
2. Alanen, M., Porres, I.: Difference and union of models. In: Stevens, P., Whittle, J., Booch, G. (eds.) Proceedings of the 6th International Conference on the UML. Lecture Notes in Computer Science, vol. **2863**, pp. 2–17. Springer (2003)
3. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. IJWIS **5**(3), 271–304 (2009)
4. Apiwattanapong, T., Orso, A., Harrold, M.J.: JDiff: A differencing technique and tool for object-oriented programs. Autom. Softw. Eng. **14**(1), 3–36 (2007)
5. Barrett, S., Chalin, P., Butler, G.: Table-driven detection and resolution of operation-based merge conflicts with mirador. In: France, R.B., Küster, J.M., Bordbar, B., Paige, R.F. (eds.) Proceedings of the 7th European Conference on Modelling Foundations and Applications (ECMFA). Lecture Notes in Computer Science, vol. **6698**, pp. 329–344. Springer (2011)
6. Briand, L.C., Labiche, Y., O'Sullivan, L., Sówka, M.M.: Automated impact analysis of UML models. J. Syst. Softw. **79**(3), 339–352 (2006)
7. Cicchetti, A., Ruscio, D.D., Pierantonio, A.: A metamodel independent approach to difference representation. J. Object Technol. **6**(9), 165–185 (2007)
8. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: there and back again. In: Proceedings of the 11th International Conference on Software Product Lines (SPLC), pp. 23–34. IEEE Computer Society (2007)
9. EMF Compare. http://www.eclipse.org/emf/compare/. Accessed 10 Aug 2016
10. Eshuis, R.: Symbolic model checking of UML activity diagrams. ACM Trans. Softw. Eng. Methodol. **15**(1), 1–38 (2006)
11. Fahrenberg, U., Acher, M., Legay, A., Wasowski, A.: Sound merging and differencing for class diagrams. In: Gnesi, S., Rensink, A. (eds.) Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE). Lecture Notes in Computer Science, vol. **8411**, pp. 63–78. Springer (2014)
12. Fahrenberg, U., Legay, A., Wasowski, A.: Vision paper: Make a difference! (semantically). In: Whittle, J., Clark, T., Kühne, T. (eds.):

Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16–21, 2011. Proceedings, Lecture Notes in Computer Science, vol. **6981**, pp. 490–500. Springer (2011)
13. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. In: Roman, G., Griswold, W.G., Nuseibeh, B. (eds.) Proceedings of the 27th International Conference on Software Engineering (ICSE), pp. 196–205. ACM (2005)
14. Gerth, C., Küster, J.M., Luckey, M., Engels, G.: Detection and resolution of conflicting change operations in version management of process models. Softw. Syst. Model. **12**(3), 517–535 (2013)
15. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? IEEE Comput. **37**(10), 64–72 (2004)
16. Jackson, D., Ladd, D.A.: Semantic Diff: A Tool for Summarizing the Effects of Modifications. In: Müller, H.A., Georges, M. (eds.) Proceedings of the International Conference on Software Maintenance (ICSM), pp. 243–252. IEEE Computer Society (1994)
17. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press, Cambridge (2006)
18. Kehrer, T., Kelter, U., Taentzer, G.: A rule-based approach to the semantic lifting of model differences in the context of model versioning. In: Alexander, P., Pasareanu, C.S., Hosking, J.G. (eds.) Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 163–172 (2011)
19. Kehrer, T., Kelter, U., Taentzer, G.: Consistency-preserving edit scripts in model versioning. In: 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 191–201 (2013)
20. Küster, J.M., Gerth, C., Engels, G.: Dependent and conflicting change operations of process models. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) Proceedings of the 5th European Conference on Model Driven Architecture- Foundations and Applications (ECMDA-FA). Lecture Notes in Computer Science, vol. **5562**, pp. 158–173. Springer (2009)
21. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: Dumas, M., Reichert, M., Shan, M.C. (eds.) Proceedings of the 6th International Conference on Business Process Management (BPM). Lecture Notes in Computer Science, vol. **5240**, pp. 244–260. Springer (2008)
22. Langer, P., Mayerhofer, T., Kappel, G.: Semantic model differencing utilizing behavioral semantics specifications. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfrán, E. (eds.) Proceedings of the 17th International Conference on Model-Driven Engineering Languages and Systems (MODELS). Lecture Notes in Computer Science, vol. **8767**, pp. 116–132. Springer (2014)
23. Lin, Y., Gray, J., Jouault, F.: DSMDiff: a differentiation tool for domain-specific models. Eur. J. Inf. Syst. **16**(4), 349–361 (2007)
24. Lippe, E., van Oosterom, N.: Operation-based merging. In: Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments (SDE), SDE 5, pp. 78–87. ACM, New York, NY, USA (1992)
25. Maoz, S., Ringert, J.O., Rumpe, B.: A manifesto for semantic model differencing. In: Dingel, J., Solberg, A. (eds.) MODELS 2010 Workshops. Lecture Notes in Computer Science, vol. **6627**, pp. 17–31. Springer (2011)
26. Maoz, S., Ringert, J.O., Rumpe, B.: ADDiff: Semantic differencing for activity diagrams. In: Gyimóthy, T., Zeller, A. (eds.) Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) and 13rd European Software Engineering Conference (ESEC), pp. 179–189. ACM (2011)
27. Maoz, S., Ringert, J.O., Rumpe, B.: An operational semantics for activity diagrams using SMV. Tech. Rep. AIB-2011-07, RWTH Aachen University (2011)
28. Maoz, S., Ringert, J.O., Rumpe, B.: CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In: Whittle, J., Clark, T., Kühne, T. (eds.): Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16–21, 2011. Proceedings, Lecture Notes in Computer Science, vol. **6981**, pp. 592–607. Springer (2011)
29. Maoz, S., Ringert, J.O., Rumpe, B.: CDDiff: Semantic differencing for class diagrams. In: Mezini, M. (ed.) Proceedings of the 25th European Conference on Object Oriented Programming (ECOOP). Lecture Notes in Computer Science, vol. **6813**, pp. 230–254. Springer (2011)
30. Maoz, S., Ringert, J.O., Rumpe, B.: Summarizing semantic model differences. In: Models and Evolution Workshop at MODELS (2011)
31. Maoz, S., Ringert, J.O., Rumpe, B.: Verifying component and connector models against crosscutting structural views. In: Jalote, P., Briand, L.C., van der Hoek, A. (eds.) Proceedings of the 36th International Conference on Software Engineering (ICSE), pp. 95–105. ACM (2014)
32. Maoz, S., Ringert, J.O.: A framework for relating syntactic and semantic model differences. In: Lethbridge, T., Cabot, J., Egyed, A. (eds.) Proceedings of the 18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 24–33 (2015)
33. Rindt, M., Kehrer, T., Kelter, U., Pietsch, P.: Rules for Edit Operations in Class Diagrams. Tech. rep., University of Siegen (2012). Available from http://pi.informatik.uni-siegen.de/qudimo/download/RiKKP2011.pdf
34. Störrle, H.: Making sense to modelers—presenting UML class model differences in prose. In: Hammoudi, S., Pires, L.F., Filipe, J., das Neves, R.C. (eds.) MODELSWARD, pp. 39–48. SciTePress (2013)
35. Taentzer, G., Ermel, C., Langer, P., Wimmer, M.: A fundamental approach to model versioning based on graph modifications: from theory to implementation. Softw. Syst. Model. **13**(1), 239–272 (2014)
36. Thüm, T., Batory, D.S., Kästner, C.: Reasoning about edits to feature models. In: Proceedings of the 31st International Conference on Software Engineering (ICSE), pp. 254–264. IEEE (2009)

Shahar Maoz is a faculty member at the School of Computer Science in Tel Aviv University. He received his Ph.D. from the Weizmann Institute of Science, Israel in 2009. From 2010 to 2012 he was post-doc research fellow in RWTH Aachen University, Germany, with a postdoctoral fellowship from the Minerva Foundation. He joined Tel Aviv University as a faculty member at the end of 2012 and has spent a sabbatical as visiting scientist at MIT CSAIL in 2015–2016. His research interests are in software engineering, specifically in the use of models and formal methods for software evolution, log analysis, and synthesis.

Jan Oliver Ringert received his computer science diploma from Technical University Braunschweig, in 2009, and the Ph.D. degree in computer science from RWTH Aachen University, in 2014. Currently, he has a post doc position at Tel Aviv University. His research interest covers software engineering, model-driven software development, and robotics. J. O. Ringert received scholarships from the German National Academic Foundation, the German Research Foundation, and the Minerva Foundation.