Exploring Scenario Exploration

Nuno Macedo, Alcino Cunha, and Tiago Guimarães

HASLab — High Assurance Software Laboratory
INESC TEC & Universidade do Minho, Braga, Portugal

Abstract. Model finders are very popular for exploring scenarios, helping users validate specifications by navigating through conforming model instances. To be practical, the semantics of such scenario exploration operations should be formally defined and, ideally, controlled by the users, so that they are able to quickly reach interesting scenarios.
This paper explores the landscape of scenario exploration operations, by formalizing them with a relational model finder. Several scenario exploration operations provided by existing tools are formalized, and new ones are proposed, namely to allow the user to easily explore very similar (or different) scenarios, by attaching preferences to model elements. As a proof-of-concept, such operations were implemented in the popular Alloy Analyzer, further increasing its usefulness for (user-guided) scenario exploration.

1 Introduction

With the ever-growing adoption of model-driven engineering (MDE) practices in software development, it becomes increasingly important to easily verify and validate the evolving specifications. The first step of this "debugging" process is typically *scenario exploration* [15], the generation of conforming model instances that provide quick feedback about the specifications and help to flag problems at early stages of development. *Model finders*, tools whose goal is precisely to find models that conform to given constraints, play an essential role in such tasks. Among these, Alloy [7] and its underlying model finder Kodkod [17], with a lightweight approach to formal methods, an object-oriented flavor and the ability to provide quick feedback through the generation of snapshots, have proven to be well-suited to handle MDE tasks. This is patent in the number of techniques that have been proposed to verify and validate with such tools essential MDE artifacts, like model specifications [1,8,10,3] and model transformations [5,2]. Unfortunately, model finders like Alloy have limited usefulness in scenario exploration, since they do not enable the user to control the criteria through which returned instances are selected from among those rendered consistent. In fact, it is not even possible to formalize the order under which the instances are enumerated, as this is typically imposed by the underlying opaque solving procedure.

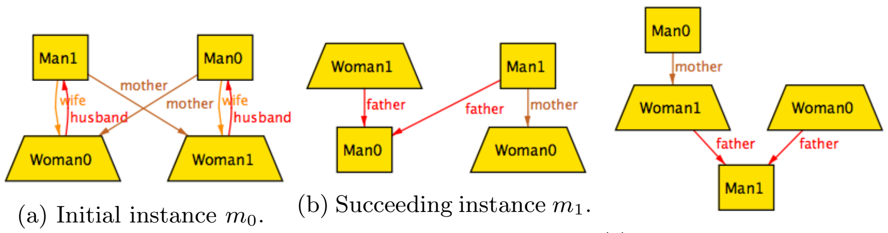
Consider, as an example, the `OwnGrandpa` Alloy module from [7], distributed with the Alloy Analyzer and depicted in Fig. 1, designed to explore the popular song "*I'm My Own Grandpa*". This module consists of `Person` elements

```
 1 abstract sig Person {
 2   father  : lone Man,
 3   mother  : lone Woman }
 4 sig Man extends Person {
 5   wife    : lone Woman }
 6 sig Woman extends Person {
 7   husband : lone Man }
 8 fact {
 9   no p:Person | p in p.^(mother+father)     // Biology
10   wife = ~husband                            // Terminology
11   no (wife+husband) & ^(mother+father)      // SocialConvention
12   Person in                                 // NoSolitary
13     Person.(mother+father+~mother+~father+wife+husband) }
14 run {} for exactly 2 Man, exactly 2 Woman
```

Fig. 1: The OwnGrandpa Alloy module.

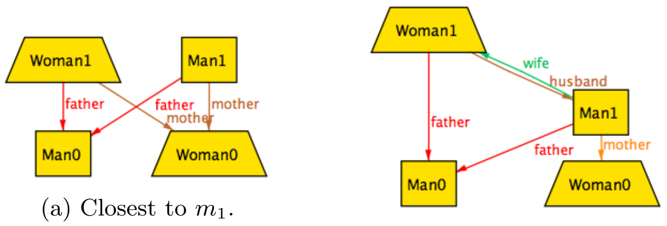
(called *atoms*, in Alloy) that are either Man or Woman, introduced by *signature* declarations (ll. 1–7). Each person may have another person with appropriate gender as father or mother, as well as a wife or husband. These relations are declared by *fields* within the signatures. An additional *fact* (ll. 8–13) imposes some restrictions on these relations: the mother and father fields must not be cyclic (Biology), wife and husband are each others inverse (Terminology), no one is married to an ancestor (SocialConvention) and there are no solitary persons (NoSolitary, not present in the original version but introduced to provide richer exploration scenarios). To validate this module and identify potential problems, the user is able to generate model instances that conform to it. The *run* command (l. 14) instructs the Analyzer to find instances with exactly 2 men and 2 women. Once an initial model is calculated, the user is able to iterate through others until there are no more valid instances left. Figure 2 presents a possible sequence of returned instances (this procedure is non-deterministic but the selected ones are representative of the result of multiple executions).

It is easy to grasp that these instances show little resemblance with one another. In fact, there is no clear order in which they are produced, which hinders the usefulness of the finder in the exploration of scenarios, since it is not predictable when problematic instances have not been generated. For instance, the user could be interested in exploring variations of the family tree m_1 (Fig. 2b), in which case instances similar to the one from Fig. 3a should be calculated. Ideally, the users should be able to go even further and customize the notion of "similar instance" for each particular context. This would allow them, for instance, to ask for models close to m_1 but prioritizing changes on marital relations, resulting instead in instances resembling the one at Fig. 3b, which quickly reveals a potential problem: this version of the OwnGrandpa module allows siblings to marry each other. Finding an instance that flags this problem using regular model findering would require an arbitrary, and possibly large, number of steps.



(a) Initial instance m_0. (b) Succeeding instance m_1. (c) Succeeding instance m_2.

Fig. 2: Regular model finding results.



(a) Closest to m_1. (b) Weighted closest to m_1.

Fig. 3: Target-oriented model finding results.

This paper explores this landscape of scenario exploration operations and formalizes them over relational model finding problems. While previous research on controlled scenario exploration exists—particularly on the generation of minimal instances [9,15,6] and on allowing the user to ask for similar instances through the introduction of additional facts [15]—a systematic study of such operations is still lacking. During this exploration it also becomes clear that many interesting operations are not supported by any existing system. We advocate that a Kodkod extension with support for *weighted target-oriented model finding* provides a formalism that is sufficiently flexible to support many interesting scenario exploration operations, including those proposed in previous studies and user-parametrizable ones that allow the generation of the instances from Fig. 3. As a proof-of-concept, we instantiate and implement some of these operations in the Alloy Analyzer, further improving its capabilities as a scenario exploration tool.

Section 2 introduces the notion of weighted target-oriented model finding, essential to define interesting operations. Section 3 formalizes scenario exploration operations on top of it, which are then deployed in an extension to the Alloy Analyzer in Section 4. Section 5 compares our formalization with existing work, while Section 6 draws conclusions and points directions to future work.

2 Relational Model Finding

The model finding formalism followed in this paper is inspired by the relational model finder Kodkod [17]. While alternative formalizations could be followed, this one has proven to be simple and flexible enough to enable the formalization of interesting scenario exploration operations. In this context, model finders are deployed through the specification of *problems* $\mathcal{P}$, which establish the search space of the finder. A problem specifies a set of bounded relation variables $\mathcal{R}$, and solving it amounts to finding a binding $B : \mathcal{R} \to \mathcal{T}$ that assigns to each relation $r \in \mathcal{R}$ a tuple set $t \in \mathcal{T}$ drawn from a fixed universe $\mathcal{A}$. While relations may be of arbitrary arity (greater than 0), its binding $B(r)$ must be uniform, containing only tuples of the same arity. This arity is imposed by its bounds, which must themselves be uniform. Bindings are isomorphic to models by translating classes as unary relations (i.e., sets) and attributes and associations as binary relations.

Definition 1 (model finding). *A* model finding problem *P is a tuple $\langle \mathcal{A}, L, U, \phi \rangle$ where $\mathcal{A}$ is a universe of atoms, $L, U : \mathcal{R} \to \mathcal{T}$ assign to each relation variable $r \in \mathcal{R}$ uniform lower- and upper-bounds, with $L(r) \subseteq U(r)$, and ϕ is a formula over $\mathcal{R}$ variables. A binding $B : \mathcal{R} \to \mathcal{T}$ is a* solution *of P, denoted by $B \models P$, if ϕ holds and $B(r) \subseteq U(r) \backslash L(r)$ for any $r \in \mathcal{R}$.*

Typically, each model finding problem P has multiple solutions. Throughout this paper, we see model finding as a procedure that somehow selects a single model from the solutions of P. We denote this selection by $B \leftarrow P$.

In previous work [4] we extended model finding problems to a *target-oriented* version, that allowed a finer control over the range of solutions of a problem. Let $\Delta : (\mathcal{R} \to \mathcal{T}) \times (\mathcal{R} \to \mathcal{T}) \to \mathbb{N}_0$ be a distance function over two bindings defined as the symmetric difference between them, defined as

$$\Delta(B, B') = \sum_{r \in \mathcal{R}_B \cap \mathcal{R}_{B'}} |B(r) \ominus B'(r)|$$

where $\mathcal{R}_B$ denotes the set of relation variables bound by B. This is equivalent to measuring the graph-edit distance between two models. Equipped with this metric we can define model finding problems whose goal is to, besides producing consistent model instances, approximate, according to Δ, a given target.

Definition 2 (target-oriented model finding). *A* target-oriented model finding problem *P_T is a tuple $\langle \mathcal{A}, L, U, T, \phi \rangle$ where $\langle \mathcal{A}, L, U, \phi \rangle$ is a model finding problem P and $T : \mathcal{R} \to \mathcal{T}$ assigns targets to a set of relation variables, with $L(r) \subseteq T(r) \subseteq U(r)$ for any $r \in \mathcal{R}_T$. A binding $B : \mathcal{R} \to \mathcal{T}$ is a* solution *of P_T, denoted by $B \models P_T$, if $B \models P$ and, for any other solution $B' \models P_T$:*

$$\Delta(B, T) \leqslant \Delta(B', T)$$

If no targets are defined, this degenerates into a regular model finding problem.

Target-oriented problems do not have necessarily a single solution, but the set of acceptable solutions is in general much smaller than the equivalent target-free problem. Nonetheless, plain graph-edit distance is too rigid to allow the user to finely control the generation of solutions. Thus, in this paper we propose a new class of *weighted* model finding problems, where the user is able to assign different weights to each relation variable, and thus control modifications over their valuation. Given a weight function $w : \mathcal{R} \to \mathbb{N}_0$, let $\Delta_w : (\mathcal{R} \to \mathcal{T}) \times (\mathcal{R} \to \mathcal{T}) \to \mathbb{N}_0$ be defined as

$$\Delta_w(B, B') = \sum_{r \in \mathcal{R}_B \cap \mathcal{R}_{B'}} |w(r)(B(r) \ominus B'(r))|$$

Given such distance function, the model finder will try to minimize the overall weighted distance between the two bindings, and as a consequence prioritize modifications on relations with smaller weight.

Definition 3 (weighted target-oriented model finding). *A* weighted target-oriented model finding problem *P_w is a tuple $\langle \mathcal{A}, L, U, T, w, \phi \rangle$ where $\langle \mathcal{A}, L, U, \phi \rangle$ is a model finding problem P, $T : \mathcal{R} \to \mathcal{T}$ assigns targets to a set of relation variables, with $L(r) \subseteq T(r) \subseteq U(r)$ for any $r \in \mathcal{R}_T$ and $w : \mathcal{R} \to \mathbb{N}_0$ is a weight function with $\mathcal{R}_T \subseteq \mathcal{R}_w$. A binding $B : \mathcal{R} \to \mathcal{T}$ is a* solution *of P_w, denoted by $B \models P_w$, if $B \models P$ and, for any other solution $B' \models P$:*

$$\Delta_w(B, T) \leqslant \Delta_w(B', T)$$

Weighted problems may still have multiple solutions. If w is a constant function, such problems degenerate into regular target-oriented ones. We will denote such weight function by $\underline{1}$. If a relation has weight 0, changes on it do not count towards the update distance, meaning that these may change freely between the target and the solutions. If all relations have weight 0, all solutions will be at the same minimal distance 0 from the target, thus degenerating into a regular model finding problem.

3 Scenario Exploration Operations

Relational model finding problems provide the base over which we formalize scenario exploration operations. Two operations are typically supported: $\mathsf{init} : \mathcal{S} \to \mathcal{P}$ creates an initial model finding problem from a specification $\mathtt{S} \in \mathcal{S}$ (for instance, an Alloy module), and $\mathsf{next}{:}\mathcal{P} \times (\mathcal{R} \to \mathcal{T}) \to \mathcal{P}$, that given the previously produced instance, updates the problem to produce a succeeding solution. Each model finding problem P that results from these operations may have multiple solutions; a single one is typically drawn from them and presented to the user. This section presents various instantiations of these two operations.

For any specification S, the embedding $[\![\mathtt{S}]\!]$ derives an appropriate weighted target-oriented model finding problem $\langle \mathcal{A}_\mathtt{S}, L_\mathtt{S}, U_\mathtt{S}, \{\,\}, \underline{1}, \phi_\mathtt{S} \rangle$ (which is equivalent to a regular model finding problem $\langle \mathcal{A}_\mathtt{S}, L_\mathtt{S}, U_\mathtt{S}, \phi_\mathtt{S} \rangle$). The embedding of the module `OwnGrandpa` from Fig. 1, which will be used as a running example, results in the model finding problem presented in Fig. 4 for the scope provided in the run command, 2 men and 2 women (atom names are abbreviated for readability).

Although model finding problems act upon relation variables, sometimes it is useful to refer to the individual atoms of the universe, which requires the reification of the atoms into relations. For instance, for universe `{M0,M1,W0,W1}`, this would give rise to 4 new singleton unary relations with exact bounds:

```
{M0,M1,W0,W1}

Man     : ({M0,M1},{M0,M1})
Woman   : ({W0,W1},{W0,W1})
father  : ({},{(M0,M0),(M0,M1),(M1,M0),(M1,M1),
               (W0,M0),(W0,M1),(W1,M0),(W1,M1)})
mother  : ({},{(M0,W0),(M0,W1),(M1,W0),(M1,W1),
               (W0,W0),(W0,W1),(W1,W0),(W1,W1)})
wife    : ({},{(M0,W0),(M1,W0),(M0,W1),(M1,W1)})
husband : ({},{(W0,M0),(W1,M0),(W0,M1),(W1,M1)})

all p:Man | lone p.wife && all p:Woman | lone p.husband
all p:Man+Woman | lone p.father && lone p.mother
all p:Man+Woman | !(p in p.^(mother+father))
wife = ~husband
no ((wife+husband) & ^(mother+father))
Man+Woman in (Man+Woman).(father+mother+~father+~mother+wife+husband)
```

Fig. 4: Kodkod embedding of the OwnGrandpa problem.

```
M0 : ({M0},{M0})    W0 : ({W0},{W0})
M1 : ({M1},{M1})    W1 : ({W1},{W1})
```

Since these are bound exactly, they do not affect the performance of the solver. Throughout this section, every model finding problem is assumed to have the atoms from its universe $\mathcal{A}$ reified into relations in $\mathcal{R}$, which allow concrete B instances to be referred in its formula ϕ. Given an operator $\approx$ that compares tuple sets, formula $[B]_\approx$ compares the valuation of relation variables $\mathcal{R}$ with their binding in B. For instance, for model m_1 (Fig. 2b), this takes the shape:

```
M0 + M1           ≈ Man     &&  W0 + W1     ≈ Woman    &&
M1->M0 + W1->M0 ≈ father  &&  M1->W0      ≈ mother   &&
none->none        ≈ wife    &&  none->none ≈ husband
```

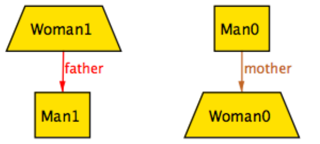
Relation **none** represents the empty set in Kodkod, and **none**->**none** the empty binary relation. Instantiating the operation with equality as $[B]_=$ results in a formula that holds iff the relations are assigned the exact same valuation as B.

3.1 Generation operations

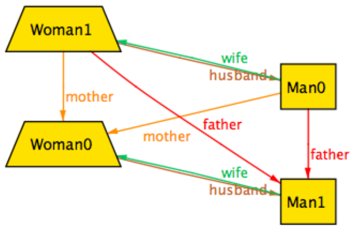
Regular generation The most basic instantiation of the generation operation, offered by Kodkod and most model finders, simply defines a problem that returns every consistent instance, amounting to a regular model finding problem:

$$\mathsf{init}(\mathsf{S}) = \langle \mathcal{A}_\mathsf{S}, L_\mathsf{S}, U_\mathsf{S}, \{\,\}, \underline{1}, \phi_\mathsf{S} \rangle$$

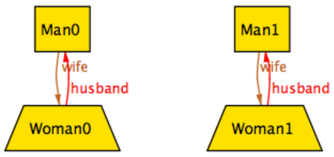
That is, the regular embedding $[\![\mathsf{S}]\!]$. Figure. 2a already presented a possible solution drawn from init(OwnGrandpa), but since the generation followed no criteria, any other model $B \models [\![\mathsf{S}]\!]$ could have been returned instead.



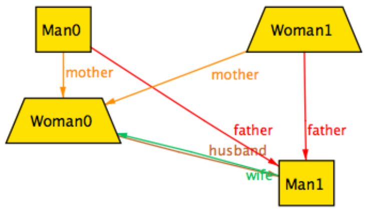
(a) $i_0 \leftarrow \mathsf{init}_\perp(\texttt{OwnGrandpa})$.



(b) $i_1 \leftarrow \mathsf{init}_\top(\texttt{OwnGrandpa})$.



(c) $i_2 \leftarrow \mathsf{init}^w_\perp(\texttt{OwnGrandpa})$.



(d) $i_3 \leftarrow \mathsf{init}_{i_1}(\texttt{OwnGrandpa}')$.

Fig. 5: Instantiations of the generation operation init.

Minimal generation Equipped with target-oriented problems, we are able to define more refined scenario exploration operations. For instance, we can define a version of init that always returns minimal solutions according to the Δ distance function, which is useful since, often, the problems of the specification are patent in even small examples. This may be achieved by trying to approximate the minimal model accepted by the problem's bounds:

$$\mathsf{init}_\perp(\texttt{S}) = \langle \mathcal{A}_\texttt{S}, L_\texttt{S}, U_\texttt{S}, L_\texttt{S}, \underline{1}, \phi_\texttt{S} \rangle$$

Note that the target cannot simply be an empty model, as that could fall below the lower bounds of $[\![\texttt{S}]\!]$. Thus, the problem should instead try to approximate $L_\texttt{S}$. For instance, in `OwnGrandpa`, the smallest potential instance amounts to:

`Man`	$\mapsto$	`{M0,M1}`
`Woman`	$\mapsto$	`{W0,W1}`
`father,mother,wife,husband`	$\mapsto$	`{}`

Figure 5a shows a solution that results from applying $\mathsf{init}_\perp$ to `OwnGrandpa`, which must relate some of its elements due to the `NoSolitary` constraint (without this constraint, there would be a single minimal solution, with the four persons unconnected). Since constraint `Terminology` forces `wife` and `husband` tuples to be paired, minimal generation will always prioritize parenthood relations.

Maximal generation A dual instantiation of minimal generation, is that of the generation of maximal solutions according to Δ, which forces the binding to assign as many tuples as possible to the relations. This is useful because it allows the exploration of boundary scenarios. Likewise to minimal generation, the largest potential instance of $[\![\texttt{S}]\!]$ is represented by the problem's upper-bound:

$$\mathsf{init}_{\top}(\mathtt{S}) = \langle \mathcal{A}_{\mathtt{S}}, L_{\mathtt{S}}, U_{\mathtt{S}}, U_{\mathtt{S}}, \underline{1}, \phi_{\mathtt{S}} \rangle$$

In `OwnGrandpa`, the largest target would take the shape:

```
Man     ↦ {M0,M1}
Woman   ↦ {W0,W1}
father  ↦ {(M0,M0),(M0,M1),(M1,M0),(M1,M1),
           (W0,M0),(W0,M1),(W1,M0),(W1,M1)}
mother  ↦ {(M0,W0),(M0,W1),(M1,W0),(M1,W1),
           (W0,W0),(W0,W1),(W1,W0),(W1,W1)}
wife    ↦ {(M0,W0),(M1,W0),(M0,W1),(M1,W1)}
husband ↦ {(W0,M0),(W1,M0),(W0,M1),(W1,M1)}
```

This results in solutions resembling the one from Fig. 5b, where problems like married siblings are immediately exposed.

Weighted generation The notion of minimality greatly varies from context to context, and may not be embodied by the simple graph-edit distance represented by Δ. Supporting weighted distances tames this strictness to a degree, allowing the user to customize the notion of minimal or maximal solution. This allows a controlled generation of boundary solutions, that can more easily trigger the detection of problems. Given a weight function w and support for weighted target-oriented model finding, this can be simply defined as:

$$\mathsf{init}^{w}_{\perp}(\mathtt{S}) = \langle \mathcal{A}_{\mathtt{S}}, L_{\mathtt{S}}, U_{\mathtt{S}}, L_{\mathtt{S}}, w, \phi_{\mathtt{S}} \rangle$$
$$\mathsf{init}^{w}_{\top}(\mathtt{S}) = \langle \mathcal{A}_{\mathtt{S}}, L_{\mathtt{S}}, U_{\mathtt{S}}, U_{\mathtt{S}}, w, \phi_{\mathtt{S}} \rangle$$

For instance, the order proposed in Section 1 for `OwnGrandpa`, that prioritizes the ancestry tree, could be embodied by the following weight function:

```
Man,Woman,father,mother ↦ 3
wife,husband            ↦ 1
```

Applying $\mathsf{init}^{w}_{\perp}$ to `OwnGrandpa` using this weight function would only yield the solutions with persons connected by marriage rather than parenthood, as depicted in Fig. 5c, because a pair of tuples `wife`/`husband` weighs less than a single parenthood tuple; if instead parenthood relations were assigned weight 2, minimal solutions would combine both parenthood and marriage connections.

Generation from instance Imagine that the developer found the problematic instance i_1 (Fig. 5b) and as a consequence modified the module to a `OwnGrandpa`$'$ version by adding a constraint that supposedly forbids siblings from marrying each other. When returning to exploration, the developer could want to search for instances that resemble i_1, to make sure that no similar problems persist. Of course in this simple scenario an assertion could be defined to check that there are no marriages between siblings, but as the complexity of the properties increases, it may be simpler to explore the solutions around the problematic instance before defining assertions. This is not possible in regular model finding, but is straight-forward in target-oriented model finding, for a binding $B : \mathcal{R} \rightarrow \mathcal{T}$ representing a pre-existing model:

$$\mathsf{init}_B(\mathsf{S}) = \langle \mathcal{A}_\mathsf{S}, L_\mathsf{S}, U_\mathsf{S}, B, \underline{1}, \phi_\mathsf{S} \rangle$$

If B is still a valid solution, it is guaranteed to be returned; otherwise the one closest to it will, allowing the user to assess whether the fix to the specification was successful. If OwnGrandpa′ is well defined, then the instance from Fig. 5d will be returned, which is the solution closest to i_1 without married siblings. This scenario hints at the application of such operation in the context of model repair, restoring the consistency of inconsistent instances through minimal updates.

3.2 Iteration operations

Regular iteration The most basic iteration operation, as provided by Kodkod and most existing model finders, generates arbitrary fresh solutions, i.e., solutions that have not been previously returned. This can be defined as:

$$\mathsf{next}(\langle \mathcal{A}, L, U, \_, \_, \phi \rangle, B_0) = \langle \mathcal{A}, L, U, \{\,\}, \underline{1}, \phi \wedge \neg\, [B_0]_= \rangle$$

Recall that $[B_0]_=$ is a formula that tests whether the relation variables have the valuation of B_0; negating it removes such instance from the search space. This basic instantiation of next returns arbitrary solutions, for instance the sequence already presented in Fig. 2.

Least-change iteration Target-oriented model finding can greatly benefit iteration operations. The most evident instantiation is to always search for instances that are close to each other. This can be easily instantiated by setting the previous instance as the target of the problem for the succeeding solution:

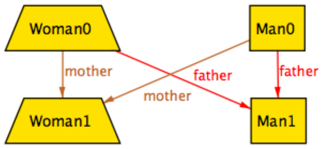
$$\mathsf{next}_\perp(\langle \mathcal{A}, L, U, \_, \_, \phi \rangle, B_0) = \langle \mathcal{A}, L, U, B_0, \underline{1}, \phi \wedge \neg\, [B_0]_= \rangle$$

Essentially, the target is now updated at each iteration to the current binding. Like the standard next operation, $\mathsf{next}_\perp$ negates the current solution in ϕ, guaranteeing that the iteration will not loop between two close instances. Figures 6a and 6b exemplify this for OwnGrandpa, assuming m_1 was returned by the preceding generation problem.

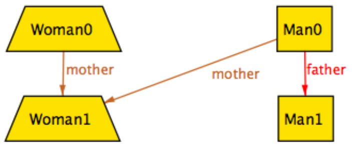
Most-change iteration In scenario exploration it is sometimes useful to iterate through solutions that greatly differ from each other, in order to explore the whole search space. Using target-oriented problems, this can be specified as:

$$\mathsf{next}_\top(\langle \mathcal{A}, L, U, \_, \_, \phi \rangle, B_0) = \langle \mathcal{A}, L, U, \overline{B_0}, \underline{1}, \phi \wedge \neg\, [B_0]_= \rangle$$

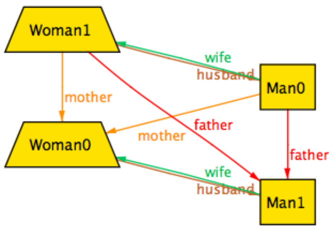
Binding $\overline{B_0}$ represents the complement of B_0 in relation to the upper-bound of the problem, i.e., for any $r \in \mathcal{R}$, $\overline{B_0}(r) = U(r) \backslash B_0(r)$, which is the potential solution further away from B_0. A possible run for OwnGrandpa is depicted in Figs. 6c and 6d, assuming m_1 as the previous solution.



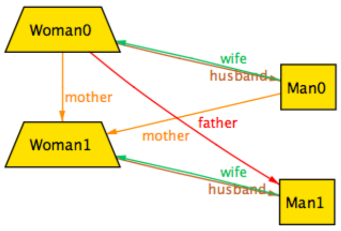
(a) $n_0 \leftarrow P_0 = \mathsf{next}_\perp(\mathsf{init}(\texttt{OwnGrandpa}), m_1)$.



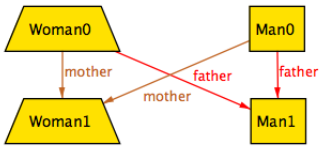
(b) $n'_0 \leftarrow \mathsf{next}_\perp(P_0, n_0)$.



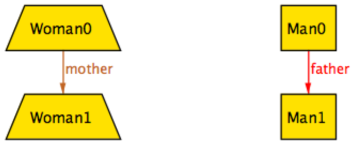
(c) $n_1 \leftarrow P_1 = \mathsf{next}_\top(\mathsf{init}(\texttt{OwnGrandpa}), m_1)$.



(d) $n'_1 \leftarrow \mathsf{next}_\top(P_1, n_1)$.



(e) $n_2 \leftarrow P_2 = \mathsf{next}_T(\mathsf{init}_{m_1}(\texttt{OwnGrandpa}), m_1)$.



(f) $n'_2 \leftarrow \mathsf{next}_T(P_2, n_2)$.

Fig. 6: Instantiations of the iteration operation next.

Circular iteration The user may find one of the returned instances interesting and wish to explore all solutions that are similar to it. Such kind of iteration that "circulates" an instance is specified by simply fixing the target of the problem:

$$\mathsf{next}_T(\langle \mathcal{A}, L, U, T, \_, \phi\rangle, B_0) = \langle \mathcal{A}, L, U, T, \underline{1}, \phi \wedge \neg\, [B_0]_=\rangle$$

Pairing this operation with $\mathsf{init}_\perp$ will iteratively produce minimal solutions by circulating the minimal model, while pairing it with init_B will enumerate all solutions that resemble model B. While not formalized, this would be the behavior embodied by the iteration operation in [4]. Figures 6e and 6f present a possible trace created by this operation for `OwnGrandpa` and m_1. As one circulates around the fixed target, solutions may begin to show little resemblance with each other.

Weighted iteration The scenario exploration operations presented hitherto allow the generation of solutions in an ordered manner, but do not allow the user to control that order. Equipped with weighted target-oriented model finding problems, the user is able to fine-tune the generation of solutions. Each of the above presented operations may be adapted to a weighted scenario, like:

$$\mathsf{next}^w_\perp(\langle \mathcal{A}, L, U, \_, \_, \phi\rangle, B_0) = \langle \mathcal{A}, L, U, B_0, w, \phi \wedge \neg\, [B_0]_=\rangle$$
$$\mathsf{next}^w_\top(\langle \mathcal{A}, L, U, \_, \_, \phi\rangle, B_0) = \langle \mathcal{A}, L, U, \overline{B_0}, w, \phi \wedge \neg\, [B_0]_=\rangle$$

For instance, once $\mathsf{init}^w_\perp$ kickstarts a problem with weights generating a minimal solution, iteration using $\mathsf{next}^w_\perp$ can be used to search for the next minimal ones.

Note that such formalization allows the weight function to be modified at each iteration. Considering the same weight function from Section 3.1 for `OwnGrandpa`, searching for close instances prioritizes changes in the marital status, while preserving the biological characteristics of the model; searching for far instances would have the opposite effect, modifying the latter as much as possible.

Cone iteration Other works on scenario exploration consider a notion of minimality that differs from ours [15]. In these, the order on solutions is defined by inclusion, i.e., a binding B is considered smaller than B' if for every $r \in \mathcal{R}$, $B(r) \subseteq B'(r)$. Minimal solutions are thus those where further removing any tuple results in an inconsistent solution. Thus, each minimal solution B has a "cone" of augmented solutions, containing instances obtained by introducing elements in B. To generate minimal solutions, the technique from [15] finds arbitrary solutions and then iteratively removes tuples until finding a minimal solutions from its cone. In our setting, this is simulated by a target-oriented problem:

$$\mathsf{next}_{\not\subseteq}(\langle \mathcal{A}, L, U, \_, \_, \phi \rangle, B_0) = (\langle \mathcal{A}, L, U, L, \underline{1}, \phi \wedge \neg\, [B_0]_{\subseteq} \rangle$$

Clause $\neg\, [B_0]_{\subseteq}$ introduced in each iteration guarantees that no more solutions from the cone of B_0 will be produced (this differs from $\neg\, [B_0]_{=}$, where B_0 will not be produced but other solutions from its cone may). The smallest target L guarantees that the minimal solution from the new cone is selected. The solutions generated by $\mathsf{init}_{\perp}$ are guaranteed to be the minimum of a cone, thus it can be used to kickstart the iteration with $\mathsf{next}_{\not\subseteq}$. In `OwnGrandpa`, this results in solutions other than the ones considered minimal by Δ: besides those similar to i_0 (Fig. 5a), connected by parenthood tuples, those connected by marriage would also be considered minimal, since removing any of those tuples would render the solutions inconsistent (in fact, it will return the same solutions as $\mathsf{init}_{\perp}^{w}$ with weight 1 and 2 for marriage and parenthood links, respectively).

Extended iteration Once a problem is being explored, it may be useful to control the generation of the next solution by introducing additional constraints without restarting the model finding procedure. For a formula ψ, this is formalized as:

$$\mathsf{next}_{\psi}(\langle \mathcal{A}, L, U, \_, \_, \phi \rangle, B_0) = \langle \mathcal{A}, L, U, B_0, \underline{1}, \phi \wedge \psi \rangle$$

This generates the solutions closest to B_0 where ψ also holds, including itself: the negation of B_0 is left out of the formula because the user may be assessing whether B_0 remains a solution with ψ. Since the atoms are assumed to be reified in the problem, this operation can also be used to perform the augmentation operation proposed in [15] through the insertion of tuples into relations. However, since their notion of minimality differs from ours, the resulting solutions could vary in certain scenarios. This operation is related to the generation from instance: next_{ψ} may be used when the user wishes to explore solutions without persisting ψ in the original specification; init_{B} must be used if the specification is effectively updated, requiring the restart of the iteration process.

4 Deployment in the Alloy Analyzer

Some of the proposed scenario exploration operations were implemented in an extension to the Alloy Analyzer. The Analyzer is built over Kodkod, which only supports regular model finding. Thus, first, Kodkod was extended to support weighted target-oriented model finding, and second, Analyzer was modified to be able to communicate with that extended version of Kodkod.

4.1 Weighted Kodkod

Regular Kodkod problems $\langle \mathcal{A}, L, U, \phi \rangle$ are solved through an embedding into boolean logic and the deployment of SAT solvers. This embedding is performed by interpreting each n-ary relation $r \in \mathcal{R}$ as a matrix with n dimensions of size $|\mathcal{A}|$. For each tuple $\langle A_{i_1}, ..., A_{i_n} \rangle$ in the lower-bound $L(r)$, the value in $r[i_1, ..., i_n]$ is set to 1; for those outside the upper-bound $U(r)$, the value in $r[i_1, ..., i_n]$ is set to 0. For each tuple $\langle A_{i_1}, ..., A_{i_n} \rangle$ in-between the bounds $(U(r) \backslash L(r))$, a new boolean variable $r_{i_1,...,i_n}$ is created, that establishes the presence of that tuple in r. Formula ϕ is then embedded by translating operations over relations to the corresponding matrix operations. If the SAT solver is able to find a valid valuation for the variables, it represents a valid model instance of the problem.

In [4] we extended this procedure to deal with targets using *maximum satisfiability* (Max-SAT) problems, whose goal is to find an assignment that maximizes the number of satisfied clauses. Since clauses emerging from constraint ϕ should be prioritized over those introduced by the targets, we relied on *partial maximum satisfiability* (PMax-SAT) problems, where two kinds of clauses, *soft* and *hard*, can be defined: the solver must satisfy all hard clauses and maximize the number of soft clauses satisfied. Hard clauses consist of those emerging from the standard Kodkod embedding of ϕ into boolean logic; soft clauses consist of a clause $r_{i_1,...,i_n}$ for every tuple $\langle A_{i_1}, ..., A_{i_n} \rangle$ in the target $T(r)$ and a clause $\neg r_{i_1,...,i_n}$ for every tuple $\langle A_{i_1}, ..., A_{i_n} \rangle$ outside the target but within the bounds $U(r) \backslash T(r)$. Maximizing the number of soft clauses satisfied amounts to finding solutions that differ as little as possible from the target.

Max-SAT solvers typically support *weighted* clauses, in which case the solver maximizes the weighted sum of satisfied clauses (in fact, hard clauses in PMax-SAT are enforced by assigning them weights greater than the weighted sum of all soft clauses). Thus, given a weight function $w : \mathcal{R} \to \mathbb{N}_0$, it is easy to deploy weighted target-oriented model finding problems. For each n-ary relation $r \in \mathcal{R}$ with $w(r) \neq 0$, for every tuple $\langle A_{i_1}, ..., A_{i_n} \rangle$, either a soft clause $r_{i_1,...,i_n}$ or $\neg r_{i_1,...,i_n}$ with weight $w(r)$ is introduced, depending on the tuple belonging to $T(r)$ or $U(r) \backslash T(r)$ respectively. The targets of relations whose weight is 0 are simply ignored, as their valuation does not affect the target-oriented procedure. We implemented such procedure on top of SAT4J (`http://www.sat4j.org`), a pure Java SAT solver that natively handles weighted PMax-SAT problems.

4.2 Scenario Exploration in Alloy

The Alloy Analyzer is built on top of Kodkod, so implementing interesting scenario exploration operations in it requires only an adaptation to the introduced weighted target-oriented Kodkod. The current prototype has support for minimal and maximal instance generation ($\mathsf{init}_\perp$ and $\mathsf{init}_\top$) as well as weighted minimal and maximal iterations ($\mathsf{next}^w_\perp$ and $\mathsf{next}^w_\top$). These are triggered by buttons introduced alongside those for issuing regular init and next commands in the Analyzer.

One of the main advantages of the Alloy Analyzer over plain Kodkod is its ability to automatically present the solutions as graphs through an embedded visualizer. Moreover, the user is able to easily customize the presentation of the solutions through the definition of *themes*. In order to provide a seamless experience to the user, we extend the theme editor to also support the assignment of weights to each signature and field (which are both translated into relations in Kodkod), which is retrieved every time the iteration commands are called. By default, all weights are set to 1, representing a regular target-oriented model finding problem. The user is able to increase them, or set them to 0, in which case the atoms corresponding to that relation are discarded from the target.

The other scenario exploration operations could have also been implemented in Alloy in a straight-forward way. Since the theme is persisted through iterations, $\mathsf{init}^w_\perp$ and $\mathsf{init}^w_\top$ could have been implemented instead of plain $\mathsf{init}_\perp$ and $\mathsf{init}_\top$; the Analyzer allows the persistence of generated solutions as XML files, which could be used to implement init_B. Extended iteration next_ψ could be deployed by allowing the specification to be updated in a controlled way during the iteration of solutions. The implementation of these operations is left as future work.

It is worth noting that, instead of reifying the atoms into the Kodkod problem, the Analyzer short-circuits the introduction of constraint $\neg\ [B_0]_=$ at each next step directly into the SAT problem where tuples can be directly referred to, avoiding the creation of additional relations. Our implementation follows the same approach on the introduction of targets into the PMax-SAT solver.

5 Related Work

A large number of techniques have been proposed for the generation of model instances from first-order logic constraints. The most relevant to this work are those following the "MACE-style" [14], where finite models are found through an embedding of problems into propositional logic and the deployment of off-the-shelf SAT solvers, of which Kodkod is a paradigmatic example. However, while techniques for the efficient generation of models abound, few allow the user to control how they are selected. Among the operations explored in Section 3, the best-studied one is the generation of minimal models. Such techniques [9,15,6] usually rely on an iterative procedure that removes elements from found solutions, until a minimal model is reached (according to the inclusion order presented in Section 3), that can be simulated by our "cone iteration".

More closely to our work is Aluminum [15], a tool built over Alloy whose focus is precisely to allow the user to guide the solver through scenario exploration.

Two exploration operations are proposed: the generation of minimal instances and their augmentation through introduction of tuples into relations. While the proposed next_ψ operation also allows the introduction of elements (since atoms are reified), our notion of distance varies from theirs, and thus the set of generated solutions may differ. Nonetheless, its behavior could be simulated by an embedding similar that of "cone iteration" $\mathsf{next}_{\not\subseteq}$, that minimizes solutions within independent cones. Thus, the proposed operations subsume those of Aluminum.

Relational model finders have been applied in model repair, where the ability to enforce least-change iteration is essential. The authors from [16] assess the suitability of Kodkod to repair inconsistencies. Given an inconsistent solution, a consistent one is found by relaxing the bounds of the original to allow the addition or removal of tuples suspected of causing the inconsistencies. However there is no control over how close the new model is to the original one and the authors do not reason on how to manage the creation of new atoms. Our $\mathsf{next}_\perp$ operation, based on previous work on target-oriented model finding [4] guarantees that the closest solution is returned. Previously, we had attempted to produce models with least-change directly over Alloy using an iterative procedure, which was applied in the context of model repair [13] and inter-model consistency management [11,12]. While less scalable, working at the Alloy level rather than at Kodkod's directly allowed for more expressive model distance functions.

6 Conclusion

In this paper we explored scenario exploration operations, formalizing them on top of weighted target-oriented model finding problems. To assess the usefulness of the defined operations in real MDE scenarios, an in-depth empirical study would be needed. At any rate, we believe that our formalism has shown to be sufficiently expressive and flexible to allow the specification of interesting operations, including those proposed in previous work.

We have extended Kodkod to support such class of problems by relying on PMax-SAT solver with weights, and implemented some of the proposed scenario exploration operations in the Alloy Analyzer as to prove their feasibility. In the future we intend to expand this set, converting the Analyzer into a fully fledged scenario exploration tool, as well as to assess the performance of this embedding. Nonetheless, [4] shows that such approach is viable for medium-sized models, which would be adequate for this kind of application. Managing the user expectations is also a concern. While the proposed operations allow the user to customize the order of generation by assigning weights to relations, the connection between the weights and the resulting order may not be completely clear. We are studying mechanisms to automatically derive those weights through user input of what exactly is considered a "close" solution.

Acknowledgments This work is co-financed by the North Portugal Regional Operational Programme (ON.2 – O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund

(ERDF), within project NORTE-07-0124-FEDER-000062. The authors thank Antónia Lopes for suggesting interesting operations regarding maximal change.

References

1. K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. On challenges of model transformation from UML to Alloy. *SoSyM*, 9:69–86, 2010.
2. F. Büttner, M. Egea, J. Cabot, and M. Gogolla. Verification of ATL transformations using transformation models and model finders. In *ICFEM'12*, volume 7635 of *LNCS*, pages 198–213. Springer, 2012.
3. A. Cunha, A. Garis, and D. Riesco. Translating between Alloy specifications and UML class diagrams annotated with OCL. *SoSyM*, 2013.
4. A. Cunha, N. Macedo, and T. Guimarães. Target oriented relational model finding. In *FASE'14*, volume 8411 of *LNCS*, pages 17–31. Springer, 2014.
5. M. Garcia. Formalization of QVT-Relations: OCL-based static semantics and Alloy-based validation. In *MDSD'08*, pages 21–30. Shaker Verlag, 2008.
6. M. Iser, C. Sinz, and M. Taghdiri. Minimizing models for tseitin-encoded SAT instances. In *SAT'13*, volume 7962 of *LNCS*, pages 224–232, 2013.
7. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, revised edition, 2012.
8. M. Kleiner, M. D. D. Fabro, and P. Albert. Model search: Formalizing and automating constraint solving in MDE platforms. In *ECMFA'10*, volume 6138 of *LNCS*, pages 173–188. Springer, 2010.
9. M. Koshimura, H. Nabeshima, H. Fujita, and R. Hasegawa. Minimal model generation with respect to an atom set. In *FTP'09*, pages 49–59, 2009.
10. M. Kuhlmann and M. Gogolla. From UML and OCL to relational logic and back. In *MoDELS'12*, volume 7590 of *LNCS*, pages 415–431. Springer, 2012.
11. N. Macedo and A. Cunha. Implementing QVT-R bidirectional model transformations using Alloy. In *FASE'13*, volume 7793 of *LNCS*, pages 297–311. Springer, 2013.
12. N. Macedo and A. Cunha. Least-change bidirectional model transformation with QVT-R and ATL. *SoSyM*, 2014. To appear.
13. N. Macedo, T. Guimarães, and A. Cunha. Model repair and transformation with Echo. In *ASE'13*, pages 694–697. IEEE, 2013.
14. W. McCune. A Davis-Putnam program and its application to finite first-order model search: quasigroup existence problem. Technical Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, IL, May 1994.
15. T. Nelson, S. Saghafi, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Aluminum: principled scenario exploration through minimality. In *ICSE'13*, pages 232–241. IEEE/ACM, 2013.
16. R. V. D. Straeten, J. P. Puissant, and T. Mens. Assessing the Kodkod model finder for resolving model inconsistencies. In *ECMFA'11*, volume 6698 of *LNCS*, pages 69–84. Springer, 2011.
17. E. Torlak and D. Jackson. Kodkod: A relational model finder. In *TACAS'07*, volume 4424 of *LNCS*, pages 632–647. Springer, 2007.