Bordeaux: A Tool for Thinking Outside the Box

Vajih Montaghami(✉) and Derek Rayside

Electrical & Computer Engineering, University of Waterloo, Waterloo, Canada
{vmontagh,drayside}@uwaterloo.ca

Abstract. One of the great features of the Alloy Analyzer is that it can produce examples illustrating the meaning of the user's model. These *inside-the-box* examples, which are formally permissible but (potentially) undesirable, help the user understand underconstraint bugs in the model. To get similar help with overconstraint bugs in the model the user needs to see examples that are desirable but formally excluded: that is, they need to see *outside-the-box* (near-miss) examples. We have developed a prototype extension of the Alloy Analyzer, named Bordeaux, that can find these examples that are near the border of what is permitted, and hence might be desirable. More generally, Bordeaux finds a pair of examples, a, c, at a minimum distance to each other, and where a satisfies model A and c satisfies model C. The primary use case described is when model C is the negation of model A, but there are also other uses for this *relative minimization*. Previous works, such as Aluminum, have focused on finding inside-the-box examples that are absolutely minimal.

1 Introduction

Examples can help people understand abstractions [1,6,22,23] such as models. One of the great features of the Alloy Analyzer is that it can mechanically generate examples of the user's model (formula). These examples are *inside-the-box*, meaning that they are consistent with the model. If the user deems the generated example desirable then it affirms that the model is a true expression of the user's intent. If the user deems the generated example undesirable, then it is a concrete representation of an *underconstraint* problem in the model: the model needs to be tightened to exclude the undesirable example. The Alloy Analyzer generates examples arbitrarily, without specifically targeting towards either desirable or undesirable examples.

If the model has a *partial overconstraint* bug, then Alloy's example generation facility is of limited assistance. A partial overconstraint bug means that the model unintentionally excludes some examples that should be included. Total overconstraint means that there are no examples that are consistent with the model. Alloy's *unsatisfiable core* feature highlights a subset of the model that causes the total overconstraint. Partial overconstraint bugs are tricky to detect [14], and currently have no explicit tool support in Alloy.

A facility for generating *near-miss examples* (*i.e.*, outside-the-box examples) might help the user diagnose partial overconstraint bugs. What the user might

M. Huisman and J. Rubin (Eds.): FASE 2017, LNCS 10202, pp. 22–39, 2017.
DOI: 10.1007/978-3-662-54494-5_2

like to see is an example that is formally excluded by the model but which she actually intends the model to include (*i.e.*, is desirable). Cognitive psychologists have found that near-miss examples revealing contrast are effective for human learning [6].

A simple, if inconvenient, technique for generating outside-the-box examples is to manually negate the model and use Alloy's existing example generation facility. But the chances of this technique generating examples that are desirable is slim, since there are typically so many more examples outside the box than inside the box. The chances of a near-miss example being desirable are higher, because a near-miss example is similar to examples that are desirable.

We have developed a technique and prototype tool, named Bordeaux, for doing *relative minimization* of examples. Given mutually inconsistent constraints, A and C, it will search for examples a and c, respectively, that are at a minimum distance to each other (measured by the number of tuples added or removed).

To find a near-miss example for A, simply set C to be the negation of A and commence the relative minimization procedure. We say that a is a *near-hit* example and that c is a *near-miss* example (both of A with respect to C). The space between the near-hit and the near-miss is the *border*: there are, by definition, no examples of either A or C on the border. Examples consistent with either A or C must be within A or C and hence are not on the border. Therefore, the distance of an example to the border cannot be assessed directly: only the distance between the near-hit and the near-miss examples can be measured.

To further guide the search towards desirable near-miss examples, the Bordeaux tool has an affordance for the user to specify which relations are permitted to differ. Bordeaux uses Alloy* [9], which is an extension of Alloy Analyzer, to solve formulas with higher-order quantifiers.

The experiments in Sect. 5 compare Bordeaux with Aluminum [13] and the Alloy Analyzer version 4.2. Bordeaux does a better job of producing pairs of near-hit and near-miss examples that are close to each other, with some computational cost. In some cases the absolute minimization technique of Aluminum produces results similar to the relative minimization technique of Bordeaux, but in other cases the results differ significantly.

Based on observations of the experiments, we design and implement two optimizations for Bordeaux in Sect. 6: scope tightening and parallelization. The key observation is that, in practice, the near-hit and near-miss are usually very close to each other. The optimizations reduce the computational cost of Bordeaux by over an order of magnitude.

In the next section, we review related works and discuss how Bordeaux differs from similar tools. Section 3 sketches an illustrative example. In Sect. 4, we define the concepts and formulas for finding near-hit and near-miss examples, and discuss some other special cases of these formulas that might be interesting for users. Section 5 demonstrates the experimental evaluation of Bordeaux and its comparison with the state-of-the-art Alloy analysis tools. Two approaches to optimize the prototype are described in Sect. 6. Section 7 concludes.

2 Related Work

Both Nelson et al. [13] and Cunha et al. [4] have proposed techniques to guide Alloy Analyzer's example generation facility towards more interesting inside examples. The stock Alloy Analyzer generates arbitrary inside examples, which might or might not be interesting, and might or might not help the user discover underconstraint bugs.

The Nelson et al. [13] extension of Alloy, called Aluminum, generates *minimal* inside examples. We say that this approach produces *absolute minimum* examples, because it finds the smallest examples that satisfy a given model. By contrast, our technique looks for *relatively minimum* pairs of examples: one inside (near-hit) and the other outside (near-miss) that are at a minimum distance from each other; they might not be absolutely minimal from Aluminum's perspective. Aluminum also has a facility for growing the minimal example, called *scenario exploration*.

Cunha et al. [4] used PMax-SAT [3] to enhance Kodkod [21] to find examples that are close to a target example. They discussed applications in data structure repair and model transformation. Perhaps this technique could be modified to replace our usage of Alloy* in Bordeaux.

When the model is completely overconstrained (*i.e.*, inconsistent), then no inside examples are possible. Shlyakhter et al. [19] enhanced Alloy to highlight the *unsatisfiable core* of such models; Torlak et al. [20] further enhanced this functionality. This tells the user a subset of the model (*i.e.*, formula) that needs to be changed, but does not give an example (because none are possible inside).

Browsing desirable outside examples might help the user understand what is wrong with the model [1]. In an empirical user study, Zayan et al. [23] evaluated the effects of using inside and outside examples in model comprehension and domain knowledge transfer. The study demonstrated evidence of the usefulness of outside in understanding the models, but did not state any preferences for particular examples. Browsing (desirable) outside examples might also help the user understand partially overconstrained models (in which some, but not all, desirable instances are possible).

Batot [2] designed a tool for automating MDE tasks. The tool generates examples from partial or complete metamodels to be evaluated or corrected by an expert. The minimality and coverage of examples are two major criteria for generating useful examples. Mottu et al. [12] proposed a mutation analysis technique to improve model transformation testing. Their technique mutates the model w.r.t. four abstract model transformation operators and generates mutants for evaluating test-suites. Macedo and Cunha [7] proposed a tool for analyzing bidirectional model transformations based on least changes using Alloy. The tool tries different number of changes to find the least number. Selecting proper scopes for Alloy Analyzer is a major obstacles to scaling the tool.

In his seminal work Winston [22], introduced using near-miss examples in learning classification procedures as well as explaining failures in learning unusual cases. Gick and Paterson [6] studied the value of near-miss examples for human learning. They found that contrasting near-miss examples were the

most effective examples for learning. Popelínsky [15] used near-miss examples for synthesizing normal logic programs from a small set of examples. Seater [18] employed the concepts of near-miss and near-hit examples to explain the role, *i.e.*, restricting or relaxing, of a constraint in a given model. Modeling By Example [8] is an unimplemented technique used to synthesize an Alloy model using near-miss and near-hit examples. The technique synthesizes an initial model from a set of examples; it learns the boundaries by generating near-miss and near-hit examples to be reviewed by the user. The near-hit and near-miss examples are from a slightly modified model.

ParAlloy [16] and Ranger [17] realize parallel analysis of models written in Alloy. Both tools partition a given Alloy model and make multiple calls to the underlying SAT-solver. The idea of parallelization in Bordeaux relies on selecting proper scopes as opposed to partitioning the model.

3 Illustrative Example

Consider a model that describes an undergraduate degree in computer engineering, as in Fig. 1. In this illustrative model, a student must take two courses to graduate, and she must have taken all necessary prerequisites for each course.

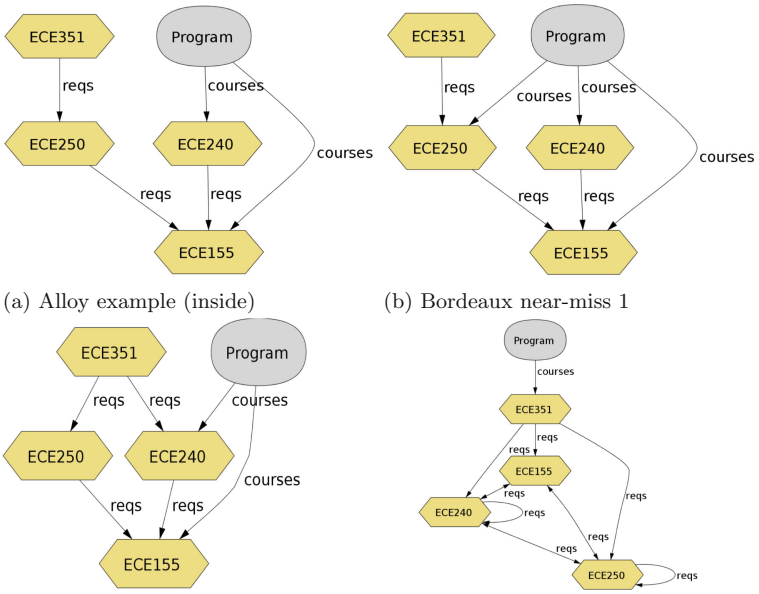
One can ask Alloy Analyzer to generate an inside example consistent with the model, the analyzer generates an example similar to Fig. 2a. Everything looks OK: this example corresponds with the user's intentions. But this model harbours a partial overconstraint bug: there are examples that the user intends, but which are not consistent with the model.

Bordeaux generates two near-miss examples (Figs. 2b and c). These are outside examples at a minimum distance from the example in Fig. 2a, adding one tuple to relations courses and reqs, respectively. The first near-miss example reveals the partial overconstraint: a student is prevented from graduating if they take an extra course. The user rectifies this by changing the equality predicate (eq[]) on Line 6 of Fig. 1 to a less-than-or-equal-to (leq[]). The second near-miss example is not interesting to the user because it just involves a perturbation of the pre-requisites. Subsequent searches can be set to exclude the reqs relation.

Alloy can be used to generate an arbitrary outside example (*e.g.*, Fig. 2d) if the user manually negates the model. This unfocused outside example is unlikely to be meaningful for the user, as it might be too divergent from her intention.

```
1 abstract sig Course{reqs: set Course}
2 one sig ECE155, ECE240, ECE250, ECE351 extends Course{}
3 one sig Program{courses: set Course}
4 pred prerequisites{ reqs = ECE240→ECE155 + ECE250→ECE155 + ECE351→ECE250 }
5 fun graduationPlan[]: Program{ {p: Program| eq[#p.courses, 2] and
6       all c: p.courses| some c.reqs implies c.reqs in p.courses} }
7 pred showSuccesfulPlan[]{  prerequisites and some graduationPlan }
8 run showSuccesfulPlan
```

Fig. 1. Model of requirements for undergraduate Computer Engineering degree



(a) Alloy example (inside)

(b) Bordeaux near-miss 1

(c) Bordeaux near-miss 2

(d) Alloy outside example

Fig. 2. Examples revealing an overconstraint issue in the model of Fig. 1

4 Proximate Pair-Finder Formula

We first define the basic concepts and formulas required to understand the formulas that Bordeaux synthesizes for producing near-hit and near-miss examples.

Definition 1 (Model). *We define an Alloy model as triple $\langle R, C, B\rangle$ comprising an ordered set of relations R, a set of constraints (formulas) on those relations C, and finite bounds B for those relations.*

Definition 2 (Valuation). *A valuation V of model M is a sequence of sets of tuples, where each entry in the sequence corresponds to a relation in M, and is within M's bounds B. Let $\mathcal{V}$ name the set of all possible valuations of M.*

The size (#) of a valuation is the number of tuples: $\#V \triangleq \sum_{i=1}^{|R|} |V_i|$

Definition 3 (Instance). *An instance I of model M is a type-correct valuation of M, according to Alloy's type system [5]. Briefly, every atom contained in the instance will be in exactly one unary relation, and the columns of each non-unary relation will be defined in terms of the unary relations.*

Suppose that I and J are two instances of model M.

The difference of I and J $(I - J)$ is a valuation of model M that, for each relation, contains the tuples from I that are not in J.

We say that J is a subset ($\subset$) of I if there is at least one relation for which J's tuples are a strict subset of I's tuples, and no relation for which I's tuples are not included in J's tuples; formally: $J \subset I \triangleq \wedge_{i=1}^{|R|}(J_i \subseteq I_i) \ \wedge \ \exists i | J_i \subset I_i$

The distance from I to J is $\#(I - J)$.

Let $\mathcal{I}$ name the set of all instances M.

Definition 4 (Inside-the-box Example). *Instance I is an inside example of model M if I satisfies M's constraints C.*

Definition 5 (Outside-the-box Example). *Instance O is an outside example of model M if O does not satisfy M's constraints C.*

4.1 Proximate Pair-Finder Formula

The core of Bordeaux generates variants of the *Proximate Pair-Finder Formula* (PPFF), which it gives to Alloy* to solve. The input to the PPFF generation is two mutually inconsistent sets of constraints, C_1 and C_2, over the same set of relations, R. A solution to the PPFF is a pair of examples, one of which (e_1) is inside C_1, and the other of which (e_2) is inside C_2. The key property of these two examples is that they are a minimum distance to each other. In the special case where C_2 is the negation of C_1, which the narrative of this paper focuses on, then e_1 is a *near-hit* example of C_1 and e_2 is a *near-miss* example of C_1.

The PPFF is expressed as a set-comprehension that returns a pair of examples e_1 and e_2. The PPFF contains two higher-order quantifiers: they are higher-order because they quantify over valuations (sets of sets). The formula effectively says that there is no other pair of examples that are closer to each other than are e_1 and e_2. Valuation v in the PPFF is the difference $e_2 - e_1$. Valuation w in the PPFF is the difference $e'_2 - e'_1$. The relative minimization condition is that the size of w is not smaller than the size of v: $\#v \leq \#w$.

In the degenerate case where C_1 and C_2 are not mutually inconsistent, then the PPFF will always return $e_1 = e_2$, because any arbitrary example is at distance zero to itself. The PPFF is not designed to be meaningful when the constraints are not mutually inconsistent.

$$
\begin{aligned}
&\{e_1, e_2 : \mathcal{I} \mid C_1[e_1] \wedge C_2[e_2] \wedge \\
&\quad \exists v : \mathcal{V} \mid (v = e_2 - e_1 \wedge e_1 \subset e_2) \wedge \\
&\qquad \forall e'_1 : \mathcal{I} - e_1,\ e'_2 : \mathcal{I} - e_2,\ w : \mathcal{V} \mid \\
&\qquad\quad (C_1[e'_1] \wedge C_2[e'_2] \wedge w = e'_2 - e'_1 \wedge e'_1 \subset e'_2) \Rightarrow \#v \leq \#w\}
\end{aligned}
$$

Fig. 3. Proximate Pair-Finder Formula (PPFF). The first line defines e_1 and e_2 as examples of C_1 and C_2, respectively. The second line defines v as the difference $e_2 - e_1$. The third line introduces alternative examples e'_1 and e'_2, and their difference w. The fourth line says that w is not less than v: *i.e.*, there is no pair of alternative examples that are closer to each other than are e_1 and e_2.

The examples e_1 and e_2 are not necessarily absolutely minimal with respect to C_1 and C_2, respectively. These two examples are *relatively* minimal with respect to each other: that is, the distance between them is small.

4.2 Encoding the PPFF for Alloy*

Alloy* supports higher-order quantifiers: *i.e.*, quantifiers over relations, which is required to solve PPFF. The user's model must be written in regular Alloy, with no higher-order quantifiers. Bordeaux transforms the user's Alloy model into an Alloy* model and adds a variant of the PPFF synthesized for the user's desired search. Bordeaux then transforms the Alloy* solution back into the terms of the user's original model.

While the Alloy* language is syntactically a superset of the regular Alloy language, so the user's model is a legal Alloy* model, simply taking the user's model as-is will not work for the PPFF. This paper focuses on the special case where C_2 is the negation of C_1, and C_1 is all of the constraints of the model. So the transformation to prepare for solving the PPFF must bundle up all of the constraints of the original model (fact blocks, multiplicity constraints, *etc.*) into a single predicate.

In actuality, the PPFF is generated using existential quantifiers rather than a set comprehension, and the skolemization gives the examples e_1 and e_2.

4.3 Special Cases of Potential User Interest

The user might be interested in some of the following special cases, which can all be easily accommodated by generating the PPFF with specific settings for C_1 and C_2 (some of these are not yet implemented in the current prototype [10]):

1. **Find a near-miss example and a near-hit example:** Set C_2 to be the negation of C_1 (as discussed above).
2. **Find a near-miss example close to an inside example:** Set C_1 to be a predicate that defines the inside example, and set C_2 to be the negation of the model's constraints.
3. **Find a near-hit example close to an outside example:** Set C_1 to be a predicate that defines the outside example, and set C_2 to be the model's constraints.
4. **Restrict the difference between the examples to certain relations:** The difference operation can easily be generated over a user-specified subset of the relations, rather than all of them.
5. **Smaller near-miss examples:** In PPFF, e_2 is bigger than e_1. If C_2 is the negation of the model's constraints, this will result in a near-miss example that is larger than the near-hit. To get a smaller near-miss example, simply set C_1 to be the negation of the model's constraints, and C_2 to be the model's constraints.

6. **Find a near-miss example for an inconsistent model:** If the original model is inconsistent, then it has no inside examples. A workaround for this situation is to set C_1 to be an empty example (no tuples), and set C_2 to be the negation of the model.

5 Experiments

To study the idea of browsing near-hit and near-miss examples, we have developed Bordeaux, a prototype that extends Alloy Analyzer. This study includes the experiments carried out to compare Bordeaux with other tools. From this study, we also show paths that optimize the performance of Bordeaux in finding near-miss examples. In this section, we explore the experiments revealing the position of Bordeaux among other similar tools. The next section discusses our ideas to optimize the prototype.

Given an example, Bordeaux can find a near-miss example. Users can browse more near-miss examples or ask for a near-hit example. To support this way of browsing, Bordeaux performs a relative minimization; namely, minimizing a distance between an inside example and an outside example. Although users cannot browse near-hit and near-miss examples with Alloy Analyzer, they can manually modify models to produce inside example and outside examples. Using Aluminum, the users can find minimal examples, and if they manually negate the model, they can browse minimal outside examples, too. Aluminum's concept of a minimal example, which we call *absolute* minimal, is an example with the smallest number of tuples.

The experiment includes five models that are shown in Table 1. We have used an Intel i7-2600K CPU at 3.40 GHz with 16 GB memory. All experiments are done with MiniSat. In what follows, we explain the experiments and discuss their contribution to answer the following research questions:

RQ-1 What is the extra cost for the relative minimums?
RQ-2 How many near-miss examples can Bordeaux find in one minute?
RQ-3a How far are arbitrary outside examples from the near-miss?
RQ-3b How far are absolute minimum outside examples from the near-miss?

To study the extra cost for finding near-miss examples with Bordeaux, we used Alloy Analyzer to find arbitrary inside examples and outside examples and compared their costs to using Bordeaux to find near-hit/near-miss example pairs (Table 1). To find the outside examples, we manually negated the studied models, *i.e.*, if C is a model's constraint, then $\neg C$ gives the negation of the model. In these experiments, for Bordeaux, we set C_1 to be equal to the arbitrary example returned by Alloy.

In Table 1, it can be seen that Bordeaux does not incur much additional cost for small models, but once the model gets larger the costs get significant (Item RQ-1). The small Binary Tree model is an exception where Bordeaux appears to run faster than the stock Alloy Analyzer. Occasional anomalies such as this are common with technology based on SAT solvers.

Table 1. Comparing Bordeaux (B) and Alloy Analyzer (A) to find outside examples

			# SAT variables			# SAT clauses			Translation time(ms)			Execution time(ms)		
	Number of relations	Size of example	B	A	B/A	B	A	B/A	B	A	B/A	B	A	B/A
Singly-linked List	2	1	846	492	*1.72*	2,518	757	*3.33*	15	26	*0.58*	29	20	*1.45*
Doubly-linked List	3	7	20,531	1,909	*10.75*	56,358	4,580	*12.31*	39,700	141	*281.56*	121,664	111	*1,096.07*
Binary Tree	3	1	1,088	710	*1.53*	3,295	1,440	*2.29*	12	438	*0.03*	44	166	*0.27*
Graduation Plan	5	8	5,934	734	*8.08*	17,846	1,276	*13.99*	381	336	*1.13*	439	74	*5.93*
File System	10	8	8,154	2,605	*3.13*	27,672	4,690	*5.90*	3,883	571	*6.80*	13,366	308	*43.40*

For answering Item RQ-2, we have done another experiment to count the number of distinct near-miss examples that Bordeaux generates in one minute. The results show how the prototype's performance degrades for the Alloy models with more relations or larger formula size. Given examples, Bordeaux produces *27*, *4*, *31*, *15*, and *9* distinct near-miss examples respectively for Singly-linked List, Doubly-linked List, Binary Tree, Graduation Plan, and File System models in one minute. The performance descends because Bordeaux reformulates and resolves the model per each distinct inside example and outside example. Bordeaux returns more near-miss examples for Singly-linked List and Binary Tree models, as the given examples of both models are fairly simpler than the others. Therefore, the near-miss examples will have relatively fewer tuples. That is, smaller near-miss examples lead to smaller and relatively simpler formulas for excluding redundant near-miss examples.

To answer Item RQ-3a and Item RQ-3b, we have performed another experiment to demonstrate how near-miss examples that Bordeaux systematically produces differ from outside examples that other tools produce from manually modified models. To do so, using various sizes of examples of different models, we evaluated their distances to outside examples that each instance-finder produces. We have selected Alloy Analyzer and Aluminum for comparing with Bordeaux. Although Alloy Analyzer and Aluminum do not provide capabilities for browsing outside examples, we have manually transformed the models and synthesized required statements.

For comparing relative minimal, absolute minimal, and arbitrary outside examples, we have used the aforementioned tools to find outside examples given arbitrary, small, medium, and large size examples. In the case of arbitrary examples, each tool finds a pair of inside example and outside example without any extra constraints on the size of examples. With restricted-size examples, all the tools have to first generate the same size examples, then generate outside examples for them. Depending on the models, the size of the examples varies from *two* to *five* tuples in small size examples and *nine* to *thirteen* tuples for the large size examples. We have recorded the size of inside examples and outside examples that each tool produces, as well as the number of tuples that should be added or removed from an example to make an example identical with its paired outside example.

As Fig. 4 shows, Aluminum generates absolute minimal inside examples and outside examples once the example size is arbitrary. It also always produces minimal outside examples regardless of the size of given examples. Alloy Analyzer generates arbitrary examples close to absolute minimal size, but the sizes of outside examples do not follow any particular pattern. Although Bordeaux produces examples in arbitrary sizes, it produces outside examples with one more tuple in all the models.

Depicted in Fig. 4, Bordeaux produces an outside example in a minimum distance from a given example. Answering Item RQ-3a, Alloy Analyzer behaves arbitrarily to produce outside examples close to the examples. The distances from examples to outside examples increase for larger examples. Answering RQ-3b,

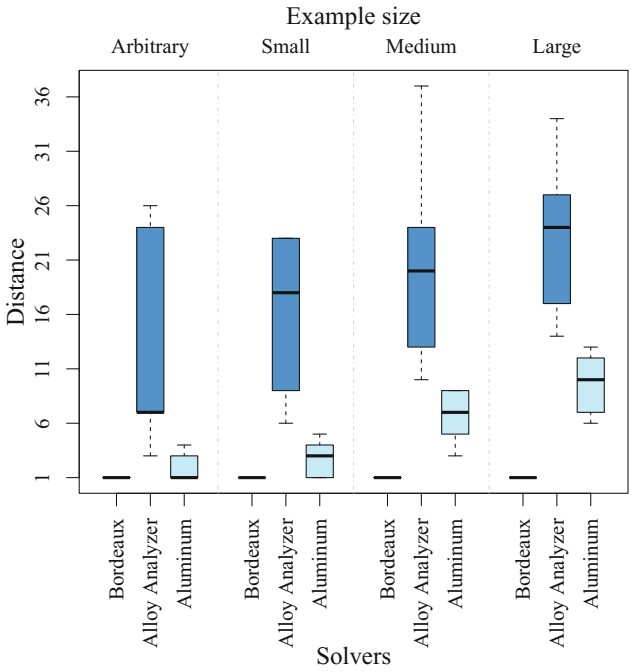


Fig. 4. Comparing Bordeaux, Alloy Analyzer, and Aluminum with respect to the number of tuples that differ between an example and an outside-the-box example.

for arbitrary and small examples, Aluminum produces outside examples that are fairly close to the examples. Given medium and large examples, Aluminum finds outside examples with larger distances from the given examples. Although the distances between inside examples and outside examples, generated by Aluminum, do not fluctuate like the distances between inside examples and outside examples produced by Alloy Analyzer, they show relative minimum distance similar to those found by Bordeaux.

Moreover, finding an outside example by negating the model provides no direction for adding or removing tuples. Although we expected to see a near-miss example with extra tuples, as generated by Bordeaux, Aluminum produced an outside example with fewer tuples for the Singly Linked-list model. Unlike Bordeaux, Alloy Analyzer and Aluminum do not directly produce outside examples of a model. Simulating a model's negation does not necessarily cause that Alloy Analyzer and Aluminum produce outside examples in a minimum distance from given examples of the studied models.

6 Optimization

By reviewing the experiment results, we have observed a trend in distances between inside examples and outside examples returned by Bordeaux. In the

studied models, with the addition of a single tuple, all inside examples and outside examples become identical. In the other words, the examples are already near-hit examples, and they can be pushed to be outside examples with the minimum number of changes, *i.e.*, a single tuple. This observation assists us to select tighter scopes and parallelize searches for inside examples and outside examples. Without choosing tight scopes, the analysis becomes infeasible. Using parallelization, the time to find near-miss examples improves to *2.2* s on average from several minutes without parallelization.

6.1 Selecting Tighter Scopes

If most examples are near-hit examples, as the case studies show, Bordeaux can approximate the scope of each unary relation to be one more than the number of its tuples in the example when Alloy* is used for the underlying solver. As depicted in Table 2, we have rerun our experimental models by selecting scopes of *one* (+1), *two* (+2), and *three* (+3) more than the number of tuples of the example for each unary relation in the models. Note that these scopes limit the number of tuples only for unary relations. Non-unary relations still can have any tuples in difference between inside example and an outside example.

When the scopes of unary relations increase by *one*, Bordeaux can find a near-miss example for a studied model within *7.5* min on average. Provided the scopes increase by two, the time to find a near-miss example is inflated by the ratio of *8.43* on average. If the scopes increase by three, the time to find a near-miss example is *fifteen* times longer than the scopes with one more unary tuple. Moreover, except for one model, Bordeaux did not terminate within *90* min if the example size is large, and the scopes increase by two. Such a lack of results within the time-limit is more frequent once the scope increases by three. Selecting the tightest scope increase can make the problem tractable for Bordeaux. If Bordeaux cannot find a near-miss example with the least scope increase, it can increase the scopes and search in a larger universe of discourse.

6.2 Parallelization

Increasing the number of atoms exponentially elevates the size of the SAT-formula, the translation time to generate it, and its solving time. In some cases, such as the Binary Tree model with a large size example, if the scope is not properly selected, Alloy* cannot find a near-miss example within several hours. Another factor that influences on the magnitude of the SAT-formula is the number of integer atoms that Bordeaux incorporates into the formula to prevent the integer overflow that might occur for distance calculations.

Observing that most examples are near-hit examples and can become near-miss examples by adding or removing a single tuple, we make new formulas so that each one applies PPFF on individual relations. Solving each formula, Bordeaux may find a near-miss example for a given example regarding a particular relation of the model.

Table 2. Showing how selecting different scopes affects the cost of analysis performed by Alloy*. The notations '*+1*', '*+2*', and '*+3*' show the records when the scopes of all unary relations in the studied models are set to one, two, and three more tuples than the number of tuples in the same relations of examples. The columns with '+1' in their headers contain the actual records. The other columns contain the increase ratios.

		SAT variables			SAT clauses			Translation time(ms)			Execution time(ms)			Total time(ms)		
		+1	+2/+1	+3/+2	+1	+2/+1	+3/+2	+1	+2/+1	+3/+2	+1	+2/+1	+3/+2	+1	+2/+1	+3/+2
Arbitrary	Singly-linked List	846	1.8842	1.5558	2,518	1.9682	1.5672	15	1.0667	1.2500	29	0.828	1.458	44	0.9091	1.3750
	Doubly-linked List	20,531	2.4871	T/O	56,358	2.7368	T/O	39,700	11.8532	T/O	121,664	1.172	T/O	161,364	3.7998	T/O
	Binary Tree	1,088	1.9743	1.6294	3,295	2.0859	1.6770	12	1.1667	1.2857	4	3.500	1.429	16	1.7500	1.3571
	Graduation Plan	5,934	1.9821	1.4840	17,846	2.0284	1.5026	381	7.4436	1.7585	439	3.146	1.547	820	5.1427	1.6891
	File System	8,154	2.1634	T/O	27,672	2.2664	T/O	3,883	12.4074	T/O	13,366	15.239	T/O	17,249	14.6013	T/O
Small	Singly-linked List	1,862	1.5397	1.6993	5,858	1.5683	1.7648	12	1.4167	1.4118	19	0.947	2.222	31	1.1290	1.8286
	Doubly-linked List	3,204	1.8121	2.0541	10,922	1.9736	2.2635	31	1.7742	2.8364	174	0.632	2.036	205	0.8049	2.3030
	Binary Tree	5,371	2.0646	2.2615	17,404	2.1485	2.3097	105	5.5429	7.3849	141	13.652	3.484	246	10.1911	4.3893
	Graduation Plan	4,342	1.7725	1.5654	13,244	1.7882	1.5503	292	7.7671	1.7637	503	5.865	2.271	795	6.5635	2.0502
	File System	2,890	1.8218	1.5470	8,910	1.9274	1.5852	60	1.0500	1.2540	138	3.732	1.450	198	2.9192	1.4291
Medium	Singly-linked List	3,537	1.8476	1.3770	10,887	1.9555	1.4716	66	4.3030	0.7007	49	3.490	2.146	115	3.9565	1.2440
	Doubly-linked List	6,175	1.9869	2.1946	20,109	2.1358	2.3287	178	5.8427	7.6663	388	1.284	57.504	566	2.7173	23.8036
	Binary Tree	13,153	2.4004	T/O	44,972	2.4702	T/O	20,804	14.7966	T/O	9,873	11.299	T/O	30,677	13.6711	T/O
	Graduation Plan	4,862	1.8375	1.5640	14,838	1.8684	1.5456	291	8.0584	1.7693	155	23.587	0.963	446	13.4552	1.2781
	File System	6,946	2.2469	T/O	21,008	2.2076	T/O	12,256	15.8204	T/O	13,174	59.415	T/O	25,430	38.4043	T/O
Large	Singly-linked List	45,668	T/O	T/O	91,495	T/O	T/O	1,040,121	T/O	T/O	260	T/O	T/O	1,040,381	T/O	T/O
	Doubly-linked List	20,549	2.4846	T/O	56,436	2.7332	T/O	35,275	14.9564	T/O	163	1.227	T/O	35,438	14.8932	T/O
	Binary Tree	24,149	T/O	T/O	79,127	T/O	T/O	1,860,944	T/O	T/O	924,176	T/O	T/O	2,785,120	T/O	T/O
	Graduation Plan	16,528	T/O	T/O	34,806	T/O	T/O	7,814	T/O	T/O	35	T/O	T/O	7,849	T/O	T/O
	File System	14,215	T/O	T/O	41,246	T/O	T/O	2,839,697	T/O	T/O	2,018,781	T/O	T/O	4,858,478	T/O	T/O

Table 3. Parallelizing PPFF can improve the efficiency of Bordeaux. Columns show the ratio of metrics measured from solving without breaking PPFF, recorded in columns labeled by '+1' in Table 2, to different approaches solving PPFF for each relation. The columns *Min-R* and *Max-R* show the improvement ratio for using parallelization. For *Min-R*, the first process finds a near-miss example, and for *Max-R*, all processes finish their searches. Columns *Seq-R* shows the differences while all processes run sequentially.

		SAT variables			SAT clauses			Translation time			Execution time			Total time		
		Min-R	Max-R	Seq-R	Min-R	Max-R	Seq-R	Min-R	Max-R	Seq-R	Min-R	Max-R	Seq-R	Min-R	Max-R	Seq-R
Arbitrary	Singly-linked List	4.43	1.25	*0.55*	5.07	0.74	*0.60*	1.67	1.50	*0.52*	5.80	4.83	*1.81*	3.14	2.75	*0.98*
	Doubly-linked List	3.24	1.89	*0.45*	3.81	0.48	*0.51*	72.58	45.74	*10.79*	10,138.67	322.72	*170.40*	288.67	129.61	*36.72*
	Binary Tree	5.04	1.27	*0.84*	5.84	0.74	*0.92*	1.33	1.20	*0.43*	1.00	1.00	*0.33*	1.23	1.14	*0.40*
	Graduation Plan	2.05	1.49	*0.28*	2.11	0.65	*0.28*	2.01	1.37	*0.26*	54.88	36.58	*13.30*	4.14	2.83	*0.55*
	File System	2.98	2.22	*0.27*	3.28	0.41	*0.30*	23.25	14.93	*1.96*	1,336.60	636.48	*230.45*	97.45	61.38	*8.44*
Small	Singly-linked List	2.25	1.32	*0.51*	2.44	0.71	*0.55*	1.33	1.20	*0.43*	4.75	3.17	*1.27*	2.38	1.94	*0.72*
	Doubly-linked List	1.45	1.45	*0.34*	1.57	0.64	*0.37*	1.15	1.07	*0.24*	2.68	2.38	*1.23*	2.23	2.01	*0.76*
	Binary Tree	1.49	1.38	*0.47*	1.57	0.70	*0.49*	2.76	2.39	*0.88*	20.14	4.15	*2.82*	5.47	3.15	*1.45*
	Graduation Plan	2.36	1.67	*0.31*	2.48	0.57	*0.33*	2.25	1.36	*0.29*	83.83	71.86	*22.86*	5.85	3.58	*0.77*
	File System	5.49	2.30	*0.34*	6.27	0.40	*0.38*	1.25	1.11	*0.12*	1.05	1.02	*0.15*	1.10	1.05	*0.14*
Medium	Singly-linked List	1.95	1.18	*0.46*	2.07	0.80	*0.48*	4.40	1.53	*0.70*	9.80	6.13	*2.45*	5.75	2.25	*1.01*
	Doubly-linked List	2.19	1.49	*0.34*	2.42	0.62	*0.37*	3.71	2.66	*0.60*	5.54	4.04	*0.92*	4.80	3.47	*0.79*
	Binary Tree	3.07	2.07	*0.88*	3.34	0.46	*0.94*	73.25	44.45	*19.87*	580.76	46.79	*38.72*	101.92	45.18	*23.56*
	Graduation Plan	2.27	1.60	*0.30*	2.35	0.61	*0.31*	2.22	1.21	*0.29*	19.38	3.30	*2.25*	3.21	1.55	*0.41*
	File System	2.80	1.93	*0.23*	2.73	0.53	*0.23*	3.44	2.11	*0.27*	823.38	268.86	*75.71*	7.11	4.35	*0.55*
Large	Singly-linked List	2.52	1.31	*0.52*	3.19	0.62	*0.64*	98.98	19.26	*13.85*	8.67	4.56	*1.90*	98.73	19.24	*13.83*
	Doubly-linked List	3.23	1.88	*0.45*	3.78	0.48	*0.51*	69.71	37.89	*9.72*	7.76	0.40	*0.20*	67.24	26.45	*7.96*
	Binary Tree	2.81	1.75	*0.78*	3.16	0.55	*0.84*	312.76	182.68	*83.34*	3,513.98	337.54	*168.55*	448.27	215.48	*100.14*
	Graduation Plan	2.55	1.91	*0.35*	2.31	0.51	*0.32*	2.26	1.42	*0.31*	1.84	1.59	*0.51*	2.26	1.42	*0.31*
	File System	3.94	2.79	*0.32*	4.03	0.34	*0.34*	155.98	54.37	*9.71*	112,154.50	151.16	*149.89*	266.60	74.08	*15.88*

Finding near-miss examples for each relation has the benefit of avoiding additional integers in the universe of discourse. Depending on how many relations a model has, Bordeaux can solve a PPFF per each relation that leads to a relatively smaller universe of discourse. As Bordeaux can independently find near-miss examples per each relation, one approach is to parallelize the search so that each process searches for a near-miss example per each relation.

The parallelization applies to all relations in the model. The parallelization has no particular restriction on the scopes of the model's relations. However, selecting proper scopes increases the performance. If a process tries to find a minimum distance with respect to a unary relation, increasing the scope of the relation by one causes a found instance to be in the distance of one, provided adding tuples is requested. Since Alloy only allows restricting scope for unary relations, no increase is the tightest scopes for a process that tries to find a minimum distance with respect to a non-unary relation; more than one tuple of a non-unary relation can also change.

In this approach, if a process finds a near-miss example first that is at a distance of one from the given example, then all other processes can stop their searches. Otherwise, all processes should continue. In the end, either (a) the last process returns a near-miss example with the distance of one from the given example, (b) all processes return nothing, (c) some processes return near-miss examples with a distance of two, or (d) some processes return near-miss examples with a distance of three or more.

If PPFF can find a near-miss example in distance one from the given example, then one of the processes must be able to find it too. Clearly, the near-border examples finder formula found the example because only one relation gets or loses a tuple, so running the formula over that relation will get the same result. If such a near-miss example exists, one process has to return it. In case (a), the last finished process returns such a near-miss example.

If no process returns a near-miss example, *i.e.*, case (b), either such an instance does not exist at any distance from the given example or adding or removing more than one tuple from two or more relations turns the given example into an outside example. In the first situation, PPFF also returns no near-miss example; however, PPFF returns a near-miss example once tuples of more than a single relation need to be changed.

In case (c), some processes return near-miss examples with distance two from the given example. Then two is the true minimum distance. If there were a closer near-miss, it would be at distance one, and one of the other processes would have found it. Since that didn't happen, two is the minimum distance.

If a process returns a near-miss example with a distance of three from a given example and all other processes return no shorter distances, the PPFF might find a near-miss example in a closer distance which is exactly two. The distance might be two if simultaneously altering tuples of two relations makes a near-miss example; therefore, the individual processes cannot find such a near-miss example. If there was a near-miss example with distance one, processes should

have returned it. In this case, distance three might be a local minimum. The same argument is valid for a distance of four or more in case (d).

As the case studies show, distances between an example and its paired near-miss example is highly likely to be one; therefore, parallelizing Bordeaux would often give the near-miss examples. As discussed, the process could also provide a good approximation of the minimum distance. In our practiced cases, all near-miss examples are found in distance one from given examples.

As Table 3 shows, parallelization improves the search for near-miss examples. In all studied models, regardless of the sizes of the given examples, parallelization decreases the size of the SAT-formula, the translation time to generate it, and the solving time. We have measured this improvement by recording the time and resources taken to find the first near-miss example, as well as the time and resources taken to finish all parallel processes. Compared to using non-broken PPFF with the least scope increase, the time to concurrently find the first near-miss example decreases by the ratio of *70.9* on average. Waiting for the termination of all processes' results changes the ratio to *30.2*.

If there are not enough resources available for parallelization, sequentially running the decomposed processes still has value. The studied models show that the sum of the process times is often less than the general time when the size of an example is large. Since most of the Alloy statements synthesized for each process are the same, the translation time might be saved by reusing some parts that have already been translated for the formula of another relation. Full assessment of this idea is left for future work.

7 Conclusion

Bordeaux is a tool for finding near-hit and near-miss example pairs that are close to each other. The near-hit example is inside-the-box: it is an example of the model the user wrote. The near-miss example is outside-the-box: it is almost consistent with the user's model except for one or two crucial details. Others have found near-miss examples to be useful [1,6,22,23]. In particular, Gick and Paterson [6] found that pairing a near-miss example with a similar near-hit example increased human comprehension of the model. We posit that such pairs might be particularly helpful for discovering and diagnosing partial over-constraints in the model. Tool support for this task is currently limited.

The Bordeaux prototype has been built to work with ordinary Alloy models. It works by transforming the user's Alloy model and synthesizing a query with higher-order quantifiers that can be solved by Alloy* [9]. Through experiments we have observed that near-hit and near-miss examples often differ in no more than one tuple. We have based two optimizations on this observation: scope tightening and parallelization. Together, they significantly reduce the cost of searching. The formalization of the idea, the PPFF (Fig. 3), is more general than the specific use-case that our narrative has centred on. The formalization works from a pair of inconsistent constraints. The use-case narrative in this paper has focused on the specific circumstance when one constraint is the negation of

the other, and sometimes even more narrowly on when the first constraint is a specific example. We intend to make use of the more general facility in our forthcoming implementation of a pattern-based model debugger [11].

Acknowledgements. We thank Vijay Ganesh, Krzysztof Czarnecki, and Marsha Chechik for their helpful discussions. This work was funded in part by NSERC (National Science and Engineering Research Council of Canada).

References

1. Bak, K., Zayan, D., Czarnecki, K., Antkiewicz, M., Diskin, Z., Wasowski, A., Rayside, D.: Example-driven modeling: model = abstractions + examples. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE 2013, pp. 1273–1276. IEEE Press (2013)
2. Batot, E.: Generating examples for knowledge abstraction in MDE: a multi-objective framework. In: Balaban, M., Gogolla, M. (eds.) Proceedings of the ACM Student Research Competition at MODELS 2015 co-located with the ACM/IEEE 18th International Conference MODELS 2015, Ottawa, Canada, 29 September, 2015. CEUR Workshop Proceedings, vol. 1503, pp. 1–6 (2015). CEUR-WS.org
3. Cha, B., Iwama, K., Kambayashi, Y., Miyazaki, S.: Local search algorithms for partial maxsat. AAAI/IAAI 263268 (1997)
4. Cunha, A., Macedo, N., Guimarães, T.: Target oriented relational model finding. In: Gnesi, S., Rensink, A. (eds.) FASE 2014. LNCS, vol. 8411, pp. 17–31. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54804-8_2
5. Edwards, J., Jackson, D., Torlak, E.: A type system for object models. In: Taylor, R.N., Dwyer, M.B. (eds.) Proceedings of the 12^{th} ACM/SIGSOFT International Symposium on the Foundations of Software Engineering (FSE). Newport Beach, CA, USA, November 2004
6. Gick, M.L., Paterson, K.: Do contrasting examples facilitate schema acquisition and analogical transfer? Can. J. Psychol./Rev. Can. Psychol. **46**(4), 539 (1992)
7. Macedo, N., Cunha, A.: Implementing QVT-R bidirectional model transformations using Alloy. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 297–311. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37057-1_22
8. Mendel, L.: Modeling by example. Master's thesis, Massachusetts Institute of Technology, September 2007
9. Milicevic, A., Near, J.P., Kang, E., Jackson, D.: Alloy*: a general-purpose higher-order relational constraint solver. In: Proceedings of the 37th International Conference on Software Engineering - vol. 1, pp. 609–619. ICSE 2015, IEEE Press (2015)
10. Montaghami, V., Odunayo, O., Guntoori, B., Rayside, D.: Bordeaux prototype (2016). https://github.com/drayside/bordeaux
11. Montaghami, V., Rayside, D.: Pattern-based debugging of declarative models. In: 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 322–327. IEEE (2015)
12. Mottu, J.-M., Baudry, B., Traon, Y.: Mutation analysis testing for model transformations. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 376–390. Springer, Heidelberg (2006). doi:10.1007/11787044_28

13. Nelson, T., Saghafi, S., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Aluminum: principled scenario exploration through minimality. In: Cheng, B., Pohl, K. (eds.) Proceedings of the 35^{th} ACM/IEEE International Conference on Software Engineering (ICSE), San Francisco, CA, pp. 232–241 (2013)
14. Newcombe, C.: Debugging designs using exhaustively testable pseudo-code. Amazon Web Services (2011). Presentation Slides http://hpts.ws/papers/2011/sessions_2011/Debugging.pdf
15. Popelínský, L.: Efficient relational learning from sparse data. In: Scott, D. (ed.) AIMSA 2002. LNCS (LNAI), vol. 2443, pp. 11–20. Springer, Heidelberg (2002). doi:10.1007/3-540-46148-5_2
16. Rosner, N., Galeotti, J.P., Lopez Pombo, C.G., Frias, M.F.: ParAlloy: towards a framework for efficient parallel analysis of alloy models. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 396–397. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11811-1_33
17. Rosner, N., Siddiqui, J.H., Aguirre, N., Khurshid, S., Frias, M.F.: Ranger: parallel analysis of alloy models by range partitioning. In: 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), pp. 147–157. IEEE (2013)
18. Seater, R.M.: Core extraction and non-example generation: debugging and understanding logical models. Master's thesis, Massachusetts Institute of Technology (2004)
19. Shlyakhter, I., Seater, R., Jackson, D., Sridharan, M., Taghdiri, M.: Debugging overconstrained declarative models using unsatisfiable cores. In: Proceedings of the 18th IEEE International Conference on Automated Software Engineering, pp. 94–105. IEEE (2003)
20. Torlak, E., Chang, F.S.-H., Jackson, D.: Finding minimal unsatisfiable cores of declarative specifications. In: Cuellar, J., Maibaum, T., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 326–341. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68237-0_23
21. Torlak, E., Jackson, D.: Kodkod: a relational model finder. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 632–647. Springer, Heidelberg (2007). doi:10.1007/978-3-540-71209-1_49
22. Winston, P.H.: Artificial Intelligence, 3rd edn, pp. 150–356. Addison-Wesley, Reading (1992)
23. Zayan, D., Antkiewicz, M., Czarnecki, K.: Effects of using examples on structural model comprehension: a controlled experiment. In: Proceedings of the 36th International Conference on Software Engineering, pp. 955–966. ACM (2014)

http://www.springer.com/978-3-662-54493-8

Fundamental Approaches to Software Engineering
20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings
Huisman, M.; Rubin, J. (Eds.)
2017, XIII, 444 p. 130 illus., Softcover
ISBN: 978-3-662-54493-8