User Studies of Principled Model Finder Output

Natasha Danas[1(✉)], Tim Nelson[1(✉)], Lane Harrison[2], Shriram Krishnamurthi[1], and Daniel J. Dougherty[2]

[1] Brown University, Providence, USA
{ndanas,tn,sk}@cs.brown.edu
[2] Worcester Polytechnic Institute, Worcester, USA
{lane,dd}@cs.wpi.edu

Abstract. Model-finders such as SAT-solvers are attractive for producing concrete models, either as sample instances or as counterexamples when properties fail. However, the generated model is arbitrary. To address this, several research efforts have proposed principled forms of output from model-finders. These include minimal and maximal models, unsat cores, and proof-based provenance of facts.

While these methods enjoy elegant mathematical foundations, they have not been subjected to rigorous evaluation on users to assess their utility. This paper presents user studies of these three forms of output performed on advanced students. We find that most of the output forms fail to be effective, and in some cases even actively mislead users. To make such studies feasible to run frequently and at scale, we also show how we can pose such studies on the crowdsourcing site Mechanical Turk.

Keywords: Models · User studies · HCI · Minimization · Provenance · Unsat core

1 Introduction

Model-finding tools like SAT solvers have seen an explosive growth over the past two decades. In addition to automation, speed, and a flexible input language, they also produce concrete instances: either instances of the specification (henceforth, "spec"), or counterexamples. Therefore, they are now used either directly or indirectly to produce tools in numerous domains such as networking [20,28,33], security [2], and software engineering [21,22]. In particular, the concrete instances are valuable because they are accessible to users, such as network operators, who are not usually schooled in formal methods.

The models that these tools produce are, however, arbitrary and reflect internal algorithmic details and sometimes also probabilities. That is, the output does not follow any particular principle beyond satisfying the given spec. To counter this, many authors have proposed *principled* forms of output following well-defined mathematical properties, such as minimality [6,9,17,27,34]. Other principled output forms, like provenance [26] and unsatisfiable (henceforth, "unsat") cores [36], augment output to aid in understanding.

A. Cimatti and M. Sirjani (Eds.): SEFM 2017, LNCS 10469, pp. 168–184, 2017.
DOI: 10.1007/978-3-319-66197-1_11

These output forms have elegant mathematical properties, making them especially attractive to researchers. However, there has not been any real investigation of whether they are actually effective for users. We therefore present the first effort at evaluating these output forms. We find that they are often misleading to users, with the very properties that make them mathematically attractive causing confusion. Though our efforts are preliminary, they point to a need for the design of principled output forms to be done in conjunction with user studies: merely appealing to elegant mathematical properties for output is insufficient.

Our studies are conducted on students (Sect. 3) and on workers on a crowdsourcing platform (Sect. 4). It would be valuable to also evaluate this work with experts. Unfortunately, experts are difficult to assemble in numbers that yield statistical significance.[1] Nevertheless, as many model finders are integrated into tools (such as those cited) for end-users, advanced students and technology professionals are a reasonable proxy for (or even members of) these audiences.

Space precludes presenting the full details of our study specs; we provide full versions at http://cs.brown.edu/research/plt/dl/model-exploration-studies/, hereafter referred to as the "supplement."

2 Principled Output Methods Being Evaluated

We first describe the formalisms that this paper evaluates. Our studies use Alloy [16], a model-finder popular in the software-engineering community. Alloy searches, up to a user-specified size bound, for models that satisfy an input specification.

2.1 Minimality and Maximality

The choice of which models to present usually depends on the underlying solver algorithms. Users might be shown any models so long as they all satisfy the spec. Several authors (Sect. 5) have proposed the principle of *minimization*—an intuitively appealing notion similar to filing bug reports with only minimal test cases. In a minimal model-finder, users are shown minimal models (first). In this context, minimality is defined in terms of set containment: a model $\mathbb{M}$ is said to be smaller than another model $\mathbb{M}'$ if $\mathbb{M}$ contains a subset of what $\mathbb{M}'$ does. Note that there may be more than one minimal model. E.g., there are two different minimal models for the propositional formula $p \vee q$. A "maximal" model finder is the dual: it finds the largest models with respect to user-specified bounds on model size.

[1] We tried to conduct a study at the ABZ conference (which has exactly the expertise we need), handing out well over a hundred brief surveys on paper and electronically over several days. Sadly, we received only two responses.

2.2 UNSAT Cores

If the spec is unsatisfiable, no models can be found. A lack of models is often not sufficiently informative. Thus, some model finders return a subset of the spec that is itself unsatisfiable: an *unsat core*. This allows the user to focus on (what is often) a small portion of the spec to localize faults and refine their understanding. When a model search is actually a verification task, an unsat core represents a portion of the spec that suffices to prove the desired property up to the bounds specified. As Torlak, et al. [36] note, an unexpectedly small core can point to problems with the original property or user-specified size bounds.

2.3 Provenance

Even if models are found, each shows only what is *possible*, i.e., an example of what the spec permits. It gives no information about which model elements are necessary rather than only present due to (possibly intentional) under-constraint. Amalgam [26] is an extension of Alloy that fills this explanatory hole. For each component of a model, Amalgam can identify when that component is necessitated by other pieces of the model, along with identifying (as cores do) portions of the spec that serve in the implication.

3 Evaluation with Student Subjects

Attracting student volunteers does not appear to be easy. In a previous year, we had tried to run studies in relevant courses at both our institutions by offering students various rewards for participation. However these yielded unusably low participation rates, and it was difficult to judge the motivation of those students who did participate. Therefore, instead of seeking volunteers, this study was integrated directly into a course. This addressed our enrollment problem: out of about 70 students in the class, over 60 students participated in our studies.

Our students are from an upper-level course entitled "Logic for Systems" at Brown University. The course begins with property-based testing, leading to writing and checking specifications in Alloy. Most students are in the second half of their undergraduate education, having had numerous courses on programming, basic theory, and other topics in computer science; many also have summer internship experience in industry. A handful are graduate students (both master's and PhD). Many of the students will end up at elite companies within a year or two. As a result, though most of the students have not yet graduated, they have extensive computer science experience that is representative of many of the skills and preparation of industrial developers.

The studies were conducted at the ends of course labs; students were allowed to opt out of this part with no impact on their grade. The lab setting is useful in two ways. First, students are motivated to do the material since it is part of their course learning. (The course is not required, so students take it by choice and out of interest.) Second, students are required to attend lab, and are thus

likely to stay to perform the study. Integrating the studies into the labs meant we had certain constraints—such as the size of specifications, their placement in the semester, the number of studies, etc.—that were unavoidable. Nevertheless, we do not believe these overly limited our studies.

All the tools have been implemented and were presented as conservative extensions to Alloy. Therefore, students did not need to switch tools, use a new syntax, learn a new visualizer, etc. This eliminates many confounding variables and makes comparisons easier.

3.1 Minimality and Maximality

In this study, we evaluated how counterexample minimization helps students debug satisfiable specifications. By default, Alloy produces arbitrary models: either as concrete instances or counterexamples to help users understand why their assertion about the spec is invalid. As discussed in Sect. 2.1, minimal models are a principled output where only facts *necessary* to satisfy the specification are included (and maximality is the dual).

In lab, students first wrote a reference-counting scheme for garbage collection. Reference-counting is well-known to be sound (it never deallocates reachable memory), but it is incomplete (it can fail to deallocate unreachable memory) when the heap contains cyclic references. Teaching assistants checked that students had completed this before proceeding to the study, in which students explored counterexamples to completeness (models that contain a heap reference cycle) and were asked to propose a constraint to make the algorithm complete—in effect, by banning cycles. (The supplement provides an example spec.)

Study Design. We split the class into two experimental groups: 35 students saw only minimal counterexamples while 25 saw only maximal ones (the imbalance is an artifact of lab section sizes). We did not otherwise modify Alloy's user interface. The first minimal and maximal counterexamples are shown in Fig. 1.

We restricted students to constructing a constraint that fits the template `(all s:State | all m:HeapCell | ...)`. An ideal solution would use transitive closure, which catches cycles of *any* size:

```
all s:State | all m:HeapCell | m not in m.^(s.references)
```

Results. Our first finding was both surprising and disappointing: in both the minimal and maximal model groups, a significant proportion of participants dropped out of the study, switching back to unprincipled output (i.e., regular Alloy). We had 10 students drop out of the maximal group (leaving 15) and 28 students drop out of the minimal group (leaving 7). While we had no official complaints submitted along with the study results, many students in the lab expressed that the principled output frustrated them.

Out of the 7 remaining students in the minimal group, only 3 correctly restricted all reference cycles. 3 students incorrectly restricted self-loops only. The one other student proposed an incorrect and irrelevant edit. In retrospect

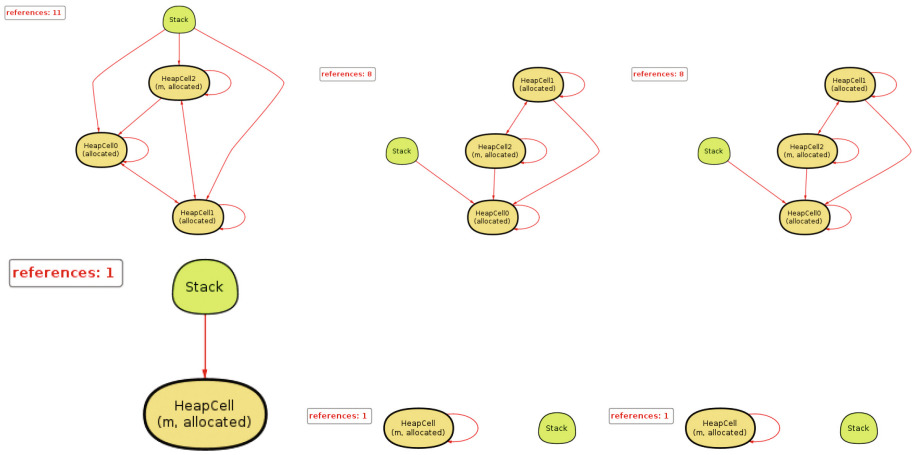

Fig. 1. Maximal (top) and minimal (bottom) counterexamples to GC completeness. Three states are shown from left to right. The transition from left to center updates the current memory references. The transition from center to right applies the specified reference-counting scheme.

this is perhaps unsurprising given the nature of models shown in minimal output, which focus attention entirely on self-loops.

In principle, maximal models do not suffer from this same tunneling of vision. Still, out of the 15 remaining maximal-group students, only 2 correctly restricted all reference cycles. 5 students incorrectly restricted self-loops only. The 8 other students proposed edits that were incorrect and irrelevant to reference cycles. Surprisingly, a higher proportion of the maximal group neglected to restrict reference cycles of any size. We discuss this and other issues in Sect. 3.4.

3.2 UNSAT Cores

In this study, we evaluated how helpful unsat cores are to students debugging unsatisfiable specs. By default, Alloy provides unsat cores to help users understand why their specification is unsatisfiable.

We presented participants with a playful, feline rendition of the "Connections of Kevin Bacon" game, where "Kitty Bacon's" connections are defined as the transitive closure of his friends. Figure 2 gives the specification in full; we explain the colored highlights below. The first group of facts (lines 1–4) define cats and how friendship works; in particular, line 3 states there is `NoSelfFriendship` allowed. Lines 6–11 define Kitty Bacon and the bounded transitive closure operator `ConnectionsOf[Cat]`. Lines 13–17 show a comparison between the bounded and unbounded notions of transitive closure. Lines 19–20 create the `CoolCatClub`, with only the connections of Kitty Bacon as members. The remainder of the specification defines the respective queries for generating cores and provenance

```
/*01*/ sig Cat {friends : set Cat}
/*02*/ fact NoFriendlessCats {no c:Cat | no c.friends}
/*03*/ fact NoSelfFriendship {no c:Cat | c in c.friends}
/*04*/ fact SymmetricFriendship {friends = ~friends}
/*05*/
/*06*/ one sig KittyBacon extends Cat {}
/*07*/ fun F[c:Cat]:set Cat {c.friends}
/*08*/ fun FF[c:Cat]:set Cat {F[F[c]]-F[c]-c}
/*09*/ fun FFF[c:Cat]:set Cat {F[F[F[c]]]-F[F[c]]-F[c]-c}
/*10*/ fun connectionsOf[c:Cat]:set Cat {F[c]+FF[c]+FFF[c]}
/*11*/ pred Connected {Cat-KittyBacon=connectionsOf[KittyBacon]}
/*12*/
/*13*/ pred SConnected {Cat-KittyBacon in KittyBacon.^friends}
/*14*/ assert IsSuperConnected {Connected iff SConnected}
/*15*/ check IsSuperConnected for exactly 3 Cat
/*16*/ check IsSuperConnected for exactly 4 Cat
/*17*/ check IsSuperConnected for exactly 5 Cat
/*18*/
/*19*/ one sig CCC {members : set Cat }
/*20*/ fact CoolCatClub {CCC.members=connectionsOf[KittyBacon]}
/*21*/
/*22*/ // UNSAT CORE QUERY
/*23*/ pred KittyBaconIsCool {KittyBacon in CCC.members}
/*24*/ run KittyBaconIsCool for exactly 4 Cat
/*25*/
/*26*/ // PROV QUERY: why not CCC.members(CCC$0,KittyBacon$0)
/*27*/ run {} for exactly 4 Cat
```

Fig. 2. Kitty Bacon spec with unsat core highlighting why Kitty Bacon is excluded (Color figure online)

(used by Sect. 3.3); here the students ran the **KittyBaconIsCool** predicate and found it was unsatisfiable—i.e., that Kitty Bacon could never be in the club.

The lab asked students to explain why the specification excluded Kitty Bacon from the set. They were shown an unsat core (the red and pink highlights in Fig. 2). We used Alloy's core minimization and granularity settings to reduce the core (i.e., the number of highlights) to its smallest size.

The core highlights the constraints responsible for unsatisfiability. The predicate being run (**KittyBaconIsCool**) fails when Kitty Bacon is excluded from the club, and the rest of the constraints together imply that he is never included. The core highlights three fragments of the specification: forbidding self-friendship (`NoSelfFriendship`), defining the connections of a cat (`ConnectionsOf[Cat]`), and the definition of club membership (`CoolCatClub`). Forbidding self-friendship means Kitty Bacon cannot be his own friend. Because he is not his own friend, Kitty Bacon is excluded from his connections. Since club membership is defined to be equivalent to the connections of Kitty Bacon, Kitty Bacon is never included in the club.

Study Design. To evaluate whether the core helped students debug their spec, we asked students to provide a free-form explanation of why Kitty Bacon was not in the club, and choose the best fix from three candidate edits. The edits

were based on the three fragments of the spec highlighted by the unsat core. The correct specification fix is to update the definition of club membership to be equivalent to the union of the connections of Kitty Bacon and Kitty Bacon himself (fixing `CoolCatClub`). This avoids changing the semantics of other predicates, which might have wider-ranging consequences. Two erroneous edits were to allow self-friendship (which violates `NoSelfFriendship`), and to add Kitty Bacon to his own connections, which invalidates the prior portion of the lab (where `ConnectionsOf[Cat]` defined bounded transitive closure). Students could optionally apply no edit if they did not know which one to choose.

Results. We split the pool of students between this study and the study of provenance (Sect. 3.3). For both groups, we code[2] the free-form explanations to match them with the candidate edits related to `NoSelfFriendship`, `ConnectionsOf[Cat]`, and `CoolCatClub`. The 28 students could blame any combination of those three[3], but could choose at most one constraint to edit.

Table 1. Effects of unsat cores on debugging Kitty Bacon spec

Constraint	# Student Blames	# Student Edits	Correct?
`CoolCatClub`	18 (64%)	22 (79%)	Y
`ConnectionsOf[Cat]`	27 (96%)	0 (0%)	N
`NoSelfFriendship`	14 (50%)	1 (4%)	N
No edit	N/A	5 (18%)	N

Table 1 shows the results. Half of the students exposed to the unsat core blamed disallowing self-friendship, but only one student applied the related (erroneous) edit. This suggests extraneous constraints in the unsat core distract students enough to widen their explanation, but not necessarily enough to cause them to apply the wrong edit. However, we constrained students to make only one change; had we permitted multiple edits, more students may have attempted erroneous ones.

3.3 Provenance

In this study, we evaluated how provenance output helps students debug a satisfiable spec (as opposed to debugging *unsatisfiable* specs aided by unsat cores). As discussed in Sect. 2.3, provenance is an alternative principled output to unsat cores, and highlights facts *necessary* to explain the presence or absence of certain tuples in an output model.

[2] Here, "coding" denotes classifying responses, not the colloquial term for programming.

[3] Only one author coded the free-form explanations into the 0–3 possible categories; thus, no inter-coder-reliability is reported. This is reasonable because the objective nature of having students give explanations along the different blame categories suggests a low likelihood of inaccurate coding.

Study Design. We had the other 35 students do the same study as in Sect. 3.2, except using provenances instead of unsat cores. The students looked at the first model returned for the specification, then asked the tool why Kitty Bacon was not in the Cool Cat Club. The tool produces two provenances. Both are subsets of the unsat core in Fig. 2. One is the same as the unsat core, except it excludes `NoSelfFriendship`. The other is the same as the previous provenance, except it excludes `ConnectionsOf[Cat]`.

Results. We code the student explanations in the same way. Again, the students could blame any combination of those three spec fragments but could only choose one edit. We expect the provenance students to blame and edit `NoSelfFriendship` less, as it is not highlighted in either provenance.

Table 2. Effects of provenance on debugging the Kitty Bacon spec

Constraint	# Student Blames	# Student Edits	Correct?
`CoolCatClub`	20 (57%)	23 (66%)	Y
`ConnectionsOf[Cat]`	21 (60%)	6 (17%)	N
`NoSelfFriendship`	9 (26%)	0 (0%)	N
No edit	N/A	6 (17%)	N

Table 2 reports our results. As expected, not highlighting `NoSelfFriendship` resulted in a only a quarter of students mentioning this constraint, and none proposing to remove it. The students who still mentioned self-friendship most likely fixated on the highlighted portions of the `ConnectionsOf[Cat]` definition that removes KittyBacon from his connected group of friends. Almost a fifth of students proposed an edit that invalidates the pedagogic portion of the lab (violating `ConnectionsOf[Cat]`). Considering that no student exposed to the unsat core proposed this erroneous edit, this result was quite surprising. A possible explanation for this surprise is discussed in Sect. 3.4.

3.4 Discussion

We hypothesize some causes for the effects that we have seen. These clearly indicate areas for future study.

Misleading Visualization. Alloy's model-visualization can impact understanding. We see several ways in which this output might have caused more maximal-group students to pick the erroneous edit; these suggest future studies. Figure 1 shows the first maximal model that students saw. Even though this model contains cycles of length 2 and 3, the immediacy and prominence of the 3 self-loops draws the eye. This may have led students in the maximal-model group to jump to the conclusion that self-loops (not cycles in general) were the problem to

be fixed. Moreover, Alloy's visualizer represents cycles of length 2 as a single, double-headed arrow. It is easy to not notice that the line represents a *pair* of (cycle-inducing) edges. In addition, the small arrowheads are easy to miss. Furthermore, self-loops and 2-cycles are *explicit*, requiring only one visual object to communicate. In contrast, cycles of size 3 and above are *implicit*; users must follow directed edges through multiple nodes to discover the cycle. This may lead to a tendency to pick out shorter cycles and miss larger ones.

Table 3. Comparing unsat core and provenance on student edits

Constraint	# Unsat Core Edits	# Provenance Edits	Correct?
`CoolCatClub`	22 (79%)	23 (66%)	Y
`ConnectionsOf[Cat]`	0 (0%)	6 (17%)	N
`NoSelfFriendship`	1 (4%)	0 (0%)	N
No edit	5 (18%)	6 (17%)	N

Unnecessary Information is Useful. The provenance output highlighted only the constraints that, *for a current model*, lead to KittyBacon's exclusion. In contrast, the unsat core output highlighted constraints that together imply KittyBacon's exclusion *for all models*. We initially expected provenance to produce higher-quality results since its output was more focused, but almost a fifth of the students exposed to provenance proposed the incorrect `ConnectionsOf[Cat]` edit—versus zero in the unsat core group (Table 3). This change invalidated bounded transitive closure from the pedagogic portion of the lab. It appears that in directing the students' attention to the extraneous `NoSelfFriendship` constraint, unsat core output helped them realize the erroneous edit would invalidate the constraint. Thus, we suspect that the "unnecessary" highlight made the students think about the problem more globally, leading to a higher-quality fix.

4 Evaluation with Crowd-Sourced Subjects

While working with classes gives us useful insights, it also imposes several constraints: we have to fit into the time the class can afford, we can run studies only when the course runs, our population size is bounded by the number of enrolled students, and so on. We would prefer larger samples to improve statistical power, and faster responses to efficiently refine our studies. In several domains, crowdsourcing has proven very useful for this purpose.

We have been trying to evaluate principled output using Amazon's Mechanical Turk (MTurk), which provides a virtually limitless population of people who will perform tasks for pay. MTurk happens to attract many technically savvy survey workers [23]; indeed, many of them fit the demographic of users of tools that employ model finders underneath.

We did not reuse the specs of Sect. 3 because we did not want to assume knowledge of garbage collection, and the Kitty Bacon spec is very intricate, making it difficult to develop concise and quick MTurk tasks. (Also, Herman et al. [15] could be read as implying that it is unwise to too directly compare formal and natural language specs, which we use on MTurk.) Instead, we used two others: one based on an address book and the other on a grade book. They are similar (but not isomorphic) in that they contain levels of indirection (the address book has aliases; the grade book has role-based access), and have constraints to prohibit erroneous configurations ("dead end" chains of address redirections; students who can both enroll in and assist grading for a course). Both are understandable to a lay person, and non-trivial while being small.

4.1 Design Decisions

Our MTurk task designs draw on research in both HCI and crowdsourcing.

Ghoniem et al. [12] show that graphs larger than twenty vertices are better represented as a matrix. We therefore avoided showing Alloy graphs. This decision was compounded by Ottley et al.'s work on multiple simultaneous representations for Bayesian reasoning tasks [30], which found the lone textual representation had most impact and, maybe even more counter-intuitively, presenting two representations at once drastically decreases performance. They also suggest presenting the problem and feedback model in a similar language, so users can easily make connections between the two. This inspired the matching syntax between the spec and model in our framework. Simons [35] remarks on participants' difficulty observing changes across sequential images. We therefore switched from multi-page, dynamically changing layouts to a single-page, static layout. We italicized relations across the specification and model because Wills [37] states that linked visualizations increase a user's chances of cognitively linking information. We used Munzner's [25] work on designing user interfaces as a general reference.

We were also inspired by the (rare) non-expert user interfaces for formal methods tools. DeOrio and Bertacco [8] pose SAT problems to humans in a way to optimize performance and engagement. Their use of shapes and highlighting to present logic puzzles to users influenced many of our visualization choices.

Kittur, Chi, and Suh [18] report on the trade-offs of sample sizes, cost, and quality on MTurk. We therefore collected meta-data in our prototypes to assess common Turker misconceptions. We paid our respondents a living wage. We also developed an informal adversarial model to filter out the remaining low quality responses; the specific number of responses removed are in line with Kittur et al.'s results on the rate of uninformative responses on MTurk.

Peer, Vosgerau, and Acquisti [31] investigate MTurk's quality controls and conclude that reputation and productivity do correlate with response quality. Therefore, we restricted our studies to Mechanical Turkers with thousands of completed tasks and high approval rates. Mason et al. [23] evaluate the differences in expert and crowdsourced populations, while also providing a blueprint

on how to properly conduct studies on Mechanical Turk. Gould et al. [13] discuss the difficulty of keeping the attention of crowdsourced subjects, finding that without intervention, crowd-workers reach inattention after about 5 min; this influenced our prototypes and final design.

4.2 Training Crowd Workers in Formal Methods

Because we cannot expect workers to know a specific formal language,[4] we systematically translate the specs to English, with a little smoothing of prose. (The full Alloy and translated specs are provided in the supplement.) For instance,

```
fact { all a: Assignment | one a.associated } ⟹
```

"Each assignment is associated with exactly one class."

We present the specs on MTurk as "logic puzzles". This hopefully attracts a more logic-minded audience, but we still want to make sure our workers understand the idea of satisfying models. We therefore include a training phase—which also serves as an assessment of the workers—before we present the actual study. This is also important for weeding out people who don't develop an understanding of the task, people clicking at random (for pay), bots, etc.

We train workers on one of the two specs, and perform the study on the full version of that spec. During training, we present the spec incrementally, adding one constraint at a time. At each step, workers are shown a collection of models and asked to classify them as satisfying or not. The non-satisfying models are not generated at random, since they might be too easy to tell apart. Rather (on steps after the first), we choose models that satisfied all but the last added constraint, and therefore "look about right". This forces workers to actually engage with the spec. (The first task's unsatisfying models are ones with type errors.)

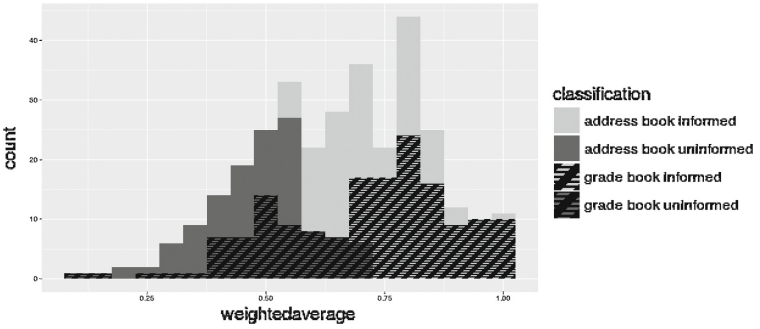

Fig. 3. Classifying crowd workers by understanding

[4] We did try to find Alloy users on MTurk. However, in twice the time it took to complete the studies of this section, we received at most 8 valid responses.

At each step, we calculate the percentage of correct classifications. At the end, we compute a weighted average of these percentages. The last step is weighted at 50%, with each previous step halving the next step's weight. This primarily weights their grade by their final classification, but also considers their earlier scores to invalidate workers who happened to guess correctly at the end. Based on the histogram of answers shown in Fig. 3, we found it useful to consider workers with a weighted average of 55% for address book and 70% for grade book. Doing so eliminated 40% and 39% of workers respectively.

We allowed all workers to proceed with the study. However, in the results presented in Sect. 4.3, we only show data from the 60% of workers who were above threshold. We analyzed the results from all workers, and found that those above threshold did perform significantly better than those below. Therefore, including all workers would result in even weaker findings below.

On MTurk, we only studied unsat cores and provenance. We did not study minimality because we presented only minimal models during the training phase. We made this choice to keep the studies small, in keeping with advice for using MTurk. Observe that Sect. 3.1 presents problems with minimality for *debugging*, not *learning*. Nevertheless, studying the use of minimality for learning remains an open question, and may require revisiting these studies.

Our MTurk studies trained 320 workers in total (192 above the threshold). The average response time was less than a minute per constraint. At a "living wage" of 15 cents a minute, grade book (with 9 constraints) would require less than $1.50 to train each respondent. We collected final results in about 6 h between two weekday mornings.

4.3 Effects of Unsat Cores and Provenance

After training, we present workers with the aforementioned erroneous configurations that the spec explicitly forbids. We generate the unsat core and provenance (shown in the supplement) for these situations similarly to the student studies. Workers are asked to blame the constraint responsible for forbidding the situation.

Table 4. Comparing proof output effects on crowd workers

Address book		Grade book	
Proof output type	# Correct	Proof output type	# Correct
Unsat core	9/49 (18%)	Unsat core	23/53 (43%)
Provenance	25/46 (54%)	Provenance	32/44 (73%)

Table 4 shows our results. For both specs, the group of workers exposed to provenance output blamed the truly responsible constraint more than those shown the unsat core. We performed a Chi-square test with Yates' continuity correction on the difference between these two proof outputs. Blaming the

proper constraint differed significantly by proof output for the address book spec ($\chi^2(1, N = 95) = 11.85$, $p < 0.001$, $\phi = 0.375$, the odds ratio is 0.28) and for the grade book spec ($\chi^2(1, N = 97) = 7.27$, $p < 0.01$, $\phi = 0.295$, the odds ratio is 2.04). As shown by Cramer's $V(\phi)$, the effect size for both specs is roughly medium (0.3).

4.4 Discussion

It is interesting that provenance is useful for MTurk workers. However, we should note three salient points. First, the specs are (intentionally) much simpler than those given to students. Second, they are working with English translations; these findings may not carry over to formal specs. Finally, these numbers only show a *relative* improvement: they may say more about the difficulty with unsat cores than about the utility of provenance.

To see the latter, we note that in address book, the provenance highlighted only *one* constraint, yet 46% did not select it! In fact, fewer workers correctly chose the single highlighted constraint of address book than between the two in grade book. It is possible that the single highlight led workers to think they were being "tricked" and made them choose a different constraint, though some free-form responses indicate this is not the case: workers genuinely intended to blame a different constraint.

In short, the studies on MTurk are very preliminary and raise many questions. Nevertheless, we believe it is worth continuing to try crowdsourcing studies to understand their limits. In particular, combining a training-and-evaluation phase with an actual evaluation task seems worth considering in future designs. Also, it may be possible, with much more time, to find several qualified Alloy users on MTurk or other platforms.

5 Related Work

Principled Model Finding. Model finders, such as Alloy [16], that rely on SAT/SMT solving techniques after converting specs into boolean logic are known as "MACE-style" model finders [24]. Koshimura et al. [19] compute minimal boolean models to solve job-scheduling problems; Aluminum [27] is a general-purpose variant of Alloy that has a similar approach. Razor [34] is a stand-alone minimal model finder that also enriches models with provenance for facts. Janota [17] generates all minimal models to aid in interactive system configuration. CPSA [9] produces minimal models specifically for the cryptographic-protocol domain. Other approaches to minimal-model generation often rely on tableaux [29] or hyper-resolution [5]. The semantics of non-monotonic reasoning [32] and database updates [10] use a more general definition of minimality. Our work has focused on evaluating Aluminum's definition of minimality; we leave exploring variations for future work. In contrast to minimality, Fu and Malik develop efficient algorithms for generating maximal models [11]. Cunha, et al. [6] implement a target-oriented model finder that generates models based

on a user-defined target metric. While general approaches like Cunha's have their own mathematical benefits, we focus on minimality (and its opposite, maximality) as a first step in user evaluation.

User Evaluation of Formal Methods Tools. Much previous work in the intersection of formal methods and HCI (e.g., much of the work appearing at the Workshop on Formal Methods in Human Computer Interaction) centers on using formal methods to improve user interfaces. These works are not significantly related to ours as we focus on the opposite: improving formal tools via user-centric evaluation.

We are not the first to use rigorous human-factors methods to evaluate formal-methods tools. Aitken, et al. [1] perform a user study to validate their hypothesis about the way experts use the HOL theorem prover. Beckert, et al. [3,4] use focus groups to detect gaps between a theorem prover's proof state and a user's internal model of the proof. Hentschel, Hähnle, and Bubel [14] evaluate two different interfaces for a program verifier and find that less experienced users performed significantly better using the interactive debugger interface. These studies all evaluate theorem proving tools, rather than a model finder. The two are fundamentally different, both in their user interfaces and in their essential function: one focuses on finding proofs, the other on constructing concrete examples. These results are therefore not directly applicable to us.

D'Antoni et al. [7] contrast the effectiveness of different feedback styles in an automata-theory tutoring program. Although their tool translates regular-language logic to English, the translation is intrinsic to part of the interface being evaluated. Our translation from Alloy to English (Sect. 4) was created solely for user evaluation and is not a component of Alloy.

All of these works evaluate *interfaces*, whereas we investigate a semantic concept: selecting which models to present. (In the case of D'Antoni et al., although counterexamples feature in the feedback, the choice of which to present is not studied.) Our work also targets a broader range of potential user backgrounds via crowdsourcing in order to obtain larger sample sizes.

6 Conclusion

Though our efforts are preliminary, they point to a need for the design of principled output forms to be done in conjunction with user studies: merely appealing to elegant mathematical properties for output is insufficient. We investigated three forms of principled output and found that, in isolation, these properties often harm user understanding. Our results suggest that minimality (and its maximal dual) can at times be frustrating and misleading, while provenance can lure users into a narrow, local perspective on their spec. While unsat cores do widen the user's vision, their full impact is not clear. User studies can help identify these unforeseen effects.

While evaluating formal methods with crowd-workers requires more effort in study design, our preliminary efforts show that it can be viable, especially

when utilizing a "train-classify-evaluate" chain of activity. Crowd-sourced user evaluations have economic, time, and sample size benefits, and hence nicely complement more in-depth, in-person studies. Additionally, effort invested into training crowd-workers may yield techniques that we can also use more broadly to educate both students and laypeople in logic and formal methods.

Acknowledgment. This work is partially supported by the US National Science Foundation.

References

1. Aitken, S., Gray, P., Melham, T., Thomas, M.: Interactive theorem proving: an empirical study of user activity. J. Symb. Comput. **25**(2), 263–284 (1998)
2. Akhawe, D., Barth, A., Lam, P., Mitchell, J., Song, D.: Towards a formal foundation of web security. In: IEEE Computer Security Foundations Symposium (2010)
3. Beckert, B., Grebing, S., Böhl, F.: How to put usability into focus: using focus groups to evaluate the usability of interactive theorem provers. In: Workshop on User Interfaces for Theorem Provers (UITP) (2014)
4. Beckert, B., Grebing, S., Böhl, F.: A usability evaluation of interactive theorem provers using focus groups. In: Workshop on Human Oriented Formal Methods (HOFM) (2014)
5. Bry, F., Yahya, A.: Positive unit hyperresolution tableaux and their application to minimal model generation. J. Autom. Reason. **25**(1), 35–82 (2000)
6. 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
7. D'Antoni, L., Kini, D., Alur, R., Gulwani, S., Viswanathan, M., Hartmann, B.: How can automatic feedback help students construct automata? Trans. Comput. Hum. Interact. **22**(2), March 2015
8. DeOrio, A., Bertacco, V.: Human computing for EDA. In: Proceedings of the 46th Annual Design Automation Conference, pp. 621–622 (2009)
9. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007). doi:10.1007/978-3-540-71209-1_41
10. Fagin, R., Ullman, J.D., Vardi, M.Y.: On the semantics of updates in databases. In: Principles of Database Systems (PODS), pp. 352–365. ACM (1983)
11. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006). doi:10.1007/11814948_25
12. Ghoniem, M., Fekete, J.D., Castagliola, P.: A comparison of the readability of graphs using node-link and matrix-based representations. In: Information Visualization (INFOVIS) (2004)
13. Gould, S., Cox, A.L., Brumby, D.P.: Diminished control in crowdsourcing: an investigation of crowdworker multitasking behavior. Trans. Comput. Hum. Interact. **23**, 19:1–19:29 (2016)
14. Hentschel, M., Hähnle, R., Bubel, R.: An empirical evaluation of two user interfaces of an interactive program verifier. In: International Conference on Automated Software Engineering (2016)

15. Herman, G.L., Kaczmarczyk, L.C., Loui, M.C., Zilles, C.B.: Proof by incomplete enumeration and other logical misconceptions. In: International Computing Education Research Workshop, ICER, pp. 59–70 (2008)
16. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press, Cambridge (2012)
17. Janota, M.: SAT solving in interactive configuration. Ph.D. thesis, University College Dublin (2010)
18. Kittur, A., Chi, E.H., Suh, B.: Crowdsourcing user studies with Mechanical Turk. In: Conference on Human Factors in Computing Systems (CHI) (2008)
19. Koshimura, M., Nabeshima, H., Fujita, H., Hasegawa, R.: Minimal model generation with respect to an atom set. In: First-Order Theorem Proving (FTP), p. 49 (2009)
20. Maldonado-Lopez, F.A., Chavarriaga, J., Donoso, Y.: Detecting network policy conflicts using Alloy. In: International Conference on Abstract State Machines, Alloy, B, and Z (2014)
21. Maoz, S., Ringert, J.O., Rumpe, B.: CD2Alloy: class diagrams analysis using Alloy revisited. In: Model Driven Engineering Languages and Systems (2011)
22. Maoz, S., Ringert, J.O., Rumpe, B.: CDDiff: semantic differencing for class diagrams. In: European Conference on Object Oriented Programming (2011)
23. Mason, W., Suri, S.: Conducting behavioral research on Amazon's Mechanical Turk. Behav. Res. Methods **44**(1), 1–23 (2012)
24. McCune, W.: Mace4 reference manual and guide. arXiv preprint cs/0310055 (2003)
25. Munzner, T.: Visualization Analysis and Design. CRC Press (2014)
26. Nelson, T., Danas, N., Dougherty, D.J., Krishnamurthi, S.: The power of "why" and "why not": enriching scenario exploration with provenance. In: Foundations of Software Engineering (2017)
27. Nelson, T., Saghafi, S., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Aluminum: principled scenario exploration through minimality. In: ICSE, pp. 232–241 (2013)
28. Nelson, T., Barratt, C., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: The Margrave tool for firewall analysis. In: Large Installation System Administration Conference (2010)
29. Niemelä, I.: A tableau calculus for minimal model reasoning. In: Miglioli, P., Moscato, U., Mundici, D., Ornaghi, M. (eds.) TABLEAUX 1996. LNCS, vol. 1071, pp. 278–294. Springer, Heidelberg (1996). doi:10.1007/3-540-61208-4_18
30. Ottley, A., Peck, E.M., Harrison, L.T., Afergan, D., Ziemkiewicz, C., Taylor, H.A., Han, P.K., Chang, R.: Improving Bayesian reasoning: the effects of phrasing, visualization, and spatial ability. Vis. Comput. Graph. **22**(1), 529–538 (2016)
31. Peer, E., Vosgerau, J., Acquisti, A.: Reputation as a sufficient condition for data quality on Amazon Mechanical Turk. Behav. Res. Methods **46**(4), 1023–1031 (2014)
32. Robinson, A., Voronkov, A.: Handbook of Automated Reasoning, vol. 1. Elsevier, Amsterdam (2001)
33. Ruchansky, N., Proserpio, D.: A (not) NICE way to verify the OpenFlow switch specification: formal modelling of the OpenFlow switch using Alloy. ACM Comput. Commun. Rev. **43**(4), 527–528 (2013)
34. Saghafi, S., Danas, R., Dougherty, D.J.: Exploring theories with a model-finding assistant. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS, vol. 9195, pp. 434–449. Springer, Cham (2015). doi:10.1007/978-3-319-21401-6_30

35. Simons, D.J.: Current approaches to change blindness. Vis. Cogn. **7**(1–3), 1–15 (2000)
36. Torlak, E., Chang, F.S.H., Jackson, D.: Finding minimal unsatisfiable cores of declarative specifications. In: International Symposium on Formal Methods (FM) (2008)
37. Wills, G.J.: Visual exploration of large structured datasets. In: Proceedings of New Techniques and Trends in Statistics (NTTS), pp. 237–246 (1997)