On Quantitative Solution Iteration in QAlloy

Pedro Silva[1,3][0000−0001−6918−5558], Nuno Macedo[2,3][0000−0002−4817−948X], and José N. Oliveira[1,3][0000−0002−0196−4229]

[1] Univ. do Minho, Braga, Portugal jno@di.uminho.pt
[2] Faculdade de Engenharia, Univ. Porto, Porto, Portugal nmacedo@fe.up.pt
[3] INESC TEC, Portugal pedro.d.silva@inesctec.pt

Abstract. A key feature of model finding techniques allows users to enumerate and explore alternative solutions. However, it is challenging to guarantee that the generated instances are relevant to the user, representing effectively different scenarios.
This challenge is exacerbated in quantitative modelling, where one must consider both the qualitative, structural part of a model, and the quantitative data on top of it. This results in a search space of possibly infinite candidate solutions, often infinitesimally similar to one another. Thus, research on instance enumeration in qualitative model finding is not directly applicable to the quantitative context, which requires more sophisticated methods to navigate the solution space effectively.
The main goal of this paper is to explore a generic approach for navigating quantitative solution spaces and showcase different iteration operations, aiming to generate instances that differ considerably from those previously seen and promote a larger coverage of the search space.
Such operations are implemented in QAlloy – a quantitative extension to Alloy – on top of Max-SMT solvers, and are evaluated against several examples ranging, in particular, over the integer and fuzzy domains.

Keywords: Quantitative Modelling · Scenario Exploration · Model Finding · Alloy · Max-SMT Solvers.

1 Introduction

Alloy [9] is a successful, lightweight formal modelling language supported by a model finding and model checking tool-set. The language finds its semantic foundations in relation algebra [26], abstracting as much as possible over concrete data so as to make model checking feasible. Such use of uninterpreted symbols is very welcome in abstract modelling but less handy wherever models involve concrete (e.g. numeric) data types behaving as measures, metrics or quantities. This led to the development of QAlloy [22,21], a quantitative extension to standard Alloy that allows reasoning about quantitative models (e.g. fuzzy).

A key feature of model discovery techniques is to allow users to explore alternative model instances. However, generating instances that are helpful to the user is challenging [4], and research has focused on providing instances that

represent truly different and illustrative scenarios [11,15,29]. This challenge becomes more difficult when considering quantitative models, where not only the qualitative and structural component of a model must be considered, but also its quantitative data.

In fact, quantities introduce a new layer of complexity in model finding because of possible infinite solution spaces. The main contributions of this paper are: *i*) the formalization of iteration operations in the context of quantitative model finding; *ii*) their implementation in QAlloy using its SMT-based backend; *iii*) an evaluation of the approach in terms of performance and diversity of generated instances.

The rest of the paper is structured as follows. Section 2 motivates the work through a couple of examples. Section 3 formalizes the iteration operations, whose implementation in QAlloy is presented in Section 4. Section 5 presents the evaluation of the approach proposed. Lastly, Section 6 presents related work, followed by conclusions and directions for future work in Section 7.

2 Quantitative Enumeration by Example

This section demonstrates the issues related with the iteration over quantitative relational instances and why previously proposed approaches for (qualitative) instance iteration are inadequate in this context. Then we briefly show how the enumeration approach proposed in this paper would result in more varied, and possibly more useful, instance exploration sessions.

We will consider two examples from distinct quantitative domains, one requiring *integer* quantities and another *fuzzy* values (reals between 0 and 1). This will help highlight the challenges that arise in domains that are inherently infinite, the former on its upper and lower bounds, and latter for the infinite number of reals in any interval. For illustration purposes we rely on QAlloy[4], although the discussion applies to quantitative model finding in general.

Example 1: Supermarket self-checkout system Consider a self-checkout system operating in a supermarket: there are *products*, that have a certain `stock` available, and *bags* processed by the system, each `contains` varying amounts of the available products. The system is expected to find inconsistencies, for instance, regarding the products in the bags and available stock. We consider a section of the supermarket handling *tea*, *coffee* and *milk* products. These are packaged in containers of varying sizes, whose *weight* is measured in *oz*, subject to some restrictions guaranteed by the suppliers. Model finding techniques are useful not only to explore scenarios that satisfy certain properties, but also to explore counterexamples that violate certain properties. Consider the verification of the constraint *"if a bag weights more than another, it means that it contains more products"*, which naturally does not hold in general. We modelled this problem in QAlloy [22], and wish to explore its counterexamples.

[4] The full QAlloy models used in this section are publicly available [20].

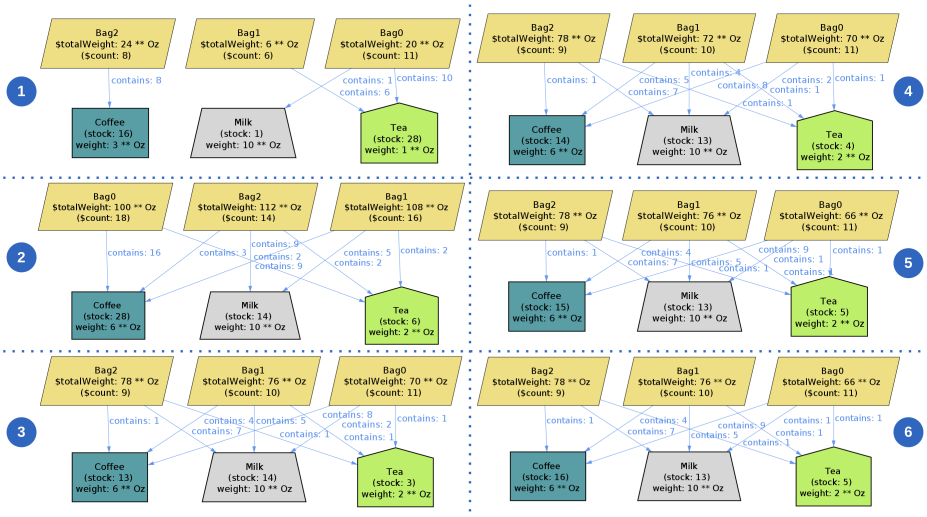

Fig. 1. Naive solution iteration on the supermarket example.

Naive iteration The most basic enumeration in model finding is to remove the current instance from the solution space by negating the current valuation of the model's variables. This is the technique used in Alloy 5 and the current QAlloy. Figure 1 shows the first counterexamples found through this approach.[5]

Two kinds of changes can be identified: in the first two steps there is a change in the *structure* of the instance, albeit a minimal change in the 2nd one. Additionally, in every step there are arbitrary changes in the quantities assigned to the model's elements.[6] After the two first steps, there are no longer any changes in the structure, and only quantities are being changed. This persists through the first 100 instances, with only occasional changes of an edge, while often getting "stuck" for dozens of instances changing the stock of a single item. These instances are valid counterexamples, but they are so similar that the user can easily miss relevant instances evidencing important issues within the model, resulting in a frustrating experience and reduced effectiveness for validation.

Separation of concerns The two kinds of change mentioned above are quite different in nature: one affects the structural aspects of the instance and is similar to traditional (qualitative) iteration; the other affects the quantities assigned to each element of the instance. Mixing these two concepts may result in modifications that may be too difficult for the user to interpret. We propose to factor iteration across two distinct axes: (a) *structure-level methods* will focus in finding instances that are structurally distinct, providing a different support on which to

[5] Recall that model finding is often backed by off-the-shelf solvers, so different solvers or configuration may result in a different sequence of instances.

[6] `$totalWeight` and `$count` are not part of the instance, but helper functions showing derived information, respectively the total weight and product count of a bag.

assign quantities, and (b) *quantity-level methods* will focus on finding alternative quantities for a fixed structure. This dichotomy will allow the user to first search for an interesting structure to examine, and then explore alternative quantities over that structure in a more predictable and controlled way; once validated, the user can ask for another structure and repeat the process. Since model finders act on a bounded domain of discourse, iteration over structures is necessarily finite and can be exhausted. This is similar in nature to the approach taken in Alloy 6 when the dynamic aspect was introduced: iteration over the static elements of the instance (its configuration) is distinct from trace iteration over a fixed configuration; likewise, the number of valid configurations is finite, but traces are potentially infinite (if no upper bound is imposed on trace prefix length).

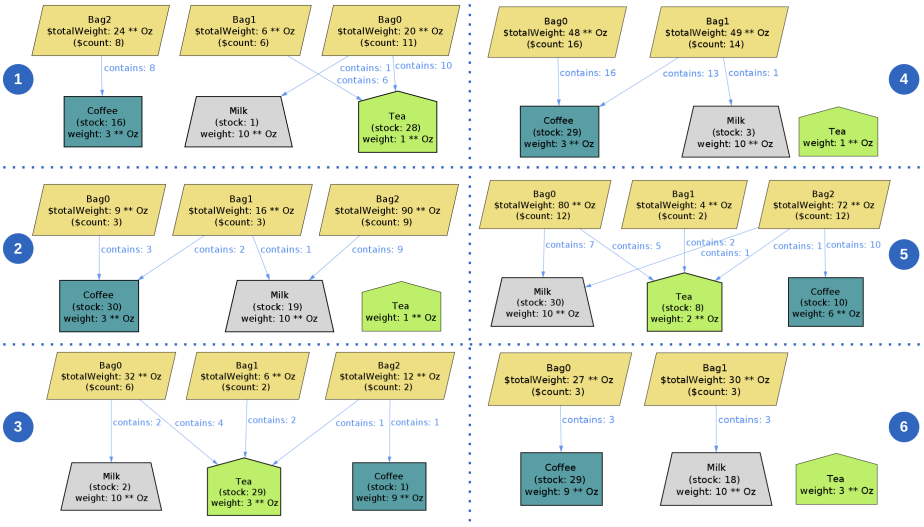

Fig. 2. Structure-level solution iteration on the supermarket example.

The sequence of solutions presented in Fig. 2 was obtained by employing an iteration approach focusing on structural changes. Notice how the instances clearly change shape, illustrating more varied scenarios, with different number of bags and contained products. Once the user finds an instance that requires further analysis, quantities can be varied for that structure. If the 3rd instance in Fig. 2 piqued the interest of the user, the iteration over quantity values only would result in the instances shown in Fig. 3. From one instance to another, there are noticeable changes between variants of the same solution structure-wise, that violate the property. For example, the 1st instance has two bags containing the same number of elements but with different weight, while in 2nd two of the bags contain more elements but weight less than a third one.

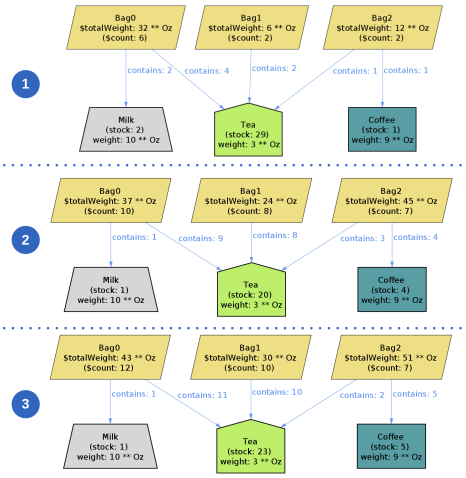

Fig. 3. Quantity-level solution iteration on the supermarket example.

Example 2: Synthesis of medical diagnoses Consider that medical professionals compile into a fuzzy relation `symptoms` the intensity with which each monitored *patient* is afflicted by each *symptom*. These patients are professionally diagnosed, the information thereof being encoded in a fuzzy relation `diagnostic` from each patient to each *disease*. A classical work on fuzzy logic [19] proposed to synthesize this information into a fuzzy relation `pathology` relating each symptom to each disease, which can then be used to produce diagnoses of new patients, through its fuzzy composition with an arbitrary `symptoms` relation. Model finding can be used to validate relation `pathology` by searching for patients with certain combinations and intensities of symptoms. Given a pre-determined relation `pathology`, suppose one wishes to explore scenarios where *"two patients do not share any common symptoms, but are diagnosed with the same diseases"*.

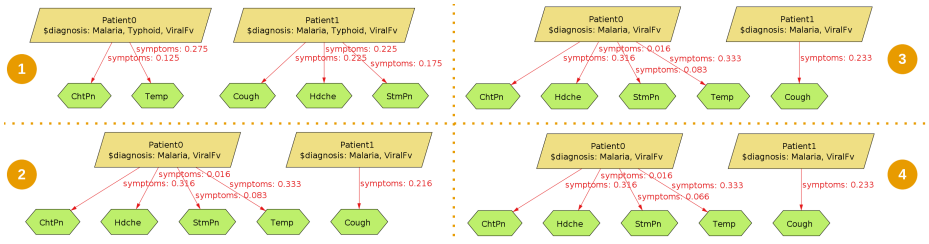

Fig. 4. Naive solution iteration for the medical diagnosis example.

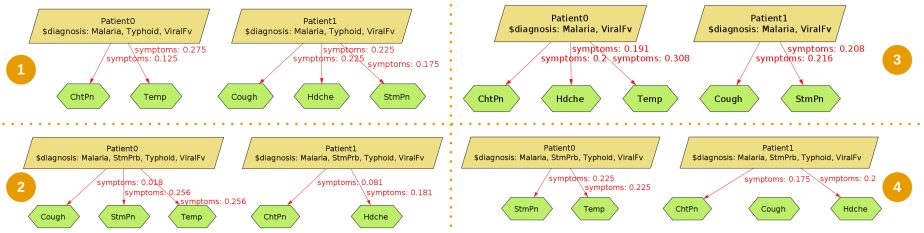

Fig. 5. Structure-level solution iteration for the medical diagnosis example.

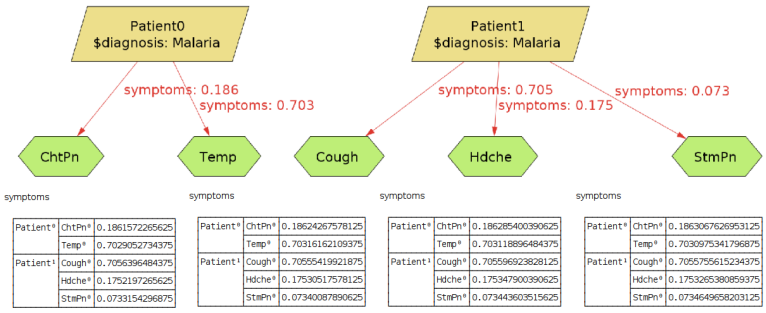

Fig. 6. Naive quantity-level solution iteration for the medical diagnosis example.

Guided iteration Having modelled this problem in QAlloy [21], Fig. 4 depicts the first instances obtained using the naive iteration approach. Similarly to example 1, only the 1st and 2nd solutions vary significantly, both patients showing the same symptoms thereafter, in very similar intensities. Therefore, the resulting diagnosis is always the same.[7] Let us apply the structure/quantity-level operations discussed previously. The former, shown in Fig. 5, leads to different combinations of symptoms under the imposed restrictions (patients with the same diagnosis). Meanwhile, quantity iterations show change in symptom intensity and a possible change in the diagnosis outcome. However, after a few more iterations, the process "converges" into the one presented in Fig. 6, producing solutions which are infinitesimally different from one another. Tabular information is shown since they are indistinguishable in the graphic representation due to rounding.

Clearly, naive quantity-level iteration is still insufficient, for such iteration operations should ensure more "vectorial distance" among instances. This problem, already explored in the qualitative setting [11], is more complex in quantitative domains: one cannot ask for instances that are furthest away from the *current* instance, because this would result in a "ping-pong" between two sets of marginally different instances. Instead, to guarantee a good coverage of the search space, one must generate instances that are far from all previously seen instances. Figure 7 displays one such enumeration, starting from the 1st solution found, from which three different diagnosis were quickly found. These varied scenarios may

[7] Relation `$diagnosis` is a derived relation selecting the most likely disease(s).

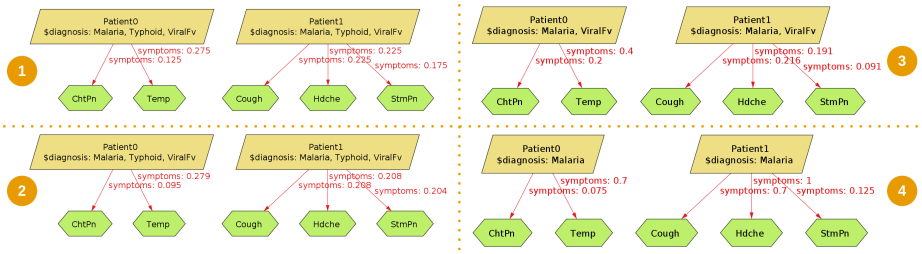

Fig. 7. Guided quantity-level solution iteration for the medical diagnosis example.

be essential for field experts to validate and refine the diagnosis relation, unlike the very similar scenarios resulting from naive iteration operations.

In summary, our main insights for this work are that *i*) iteration over structural and quantitative aspects must be provided as separate operations, and that *ii*) the latter must be guided in order to guarantee sufficiently varied instances.

3 Solution Iteration

Quantitative relational model finding A quantitative, relational model finding *problem* Q is a tuple $\langle \mathcal{M}, L, U, \phi, \mathcal{S} \rangle$, where:

- $\mathcal{M}$ is a tuple $\langle \mathbb{D}, \mathcal{U}, L_0, U_0 \rangle$ containing the static information of the problem, namely the quantitative domain (such as integers, fuzzy values, or reals) $\mathbb{D}$, the universe of atoms $\mathcal{U}$, and the original lower- and upper-bounds L_0 and U_0, which are qualitative bindings $\mathcal{R} \to \mathcal{P}(\mathcal{T}_\mathcal{U})$ where $\mathcal{T}_\mathcal{U}$ is the set of atom tuples from $\mathcal{U}$, setting the upper- and lower-bounds of free relations $\mathcal{R}$;
- $L, U : \mathcal{R} \to \mathcal{P}(\mathcal{T}_\mathcal{U})$ are the current lower- and upper-bounds such that, for any $r \in \mathcal{R}$, $L(r) \subseteq U(r)$;
- ϕ is a relational formula over variables from $\mathcal{R}$;
- $\mathcal{S}$ a set of bindings from which the instance is to be distanced.

An *instance* is a quantitative binding $I : \mathcal{R} \to \mathcal{T}_\mathcal{U} \to \mathbb{D}$ that, for each relation $r \in \mathcal{R}$, assigns a quantity from $\mathbb{D}$ to each tuple from $\mathcal{T}_\mathcal{U}$. The support of I is a qualitative binding supp $I : \mathcal{R} \to \mathcal{P}(\mathcal{T}_\mathcal{U})$, such that for each relation $r \in \mathcal{R}$, $t \in$ supp $I(r)$ iff $I(r)(t) \neq 0$. An instance I is a *solution* to a problem $Q = \langle \mathcal{M}, L, U, \phi, \mathcal{S} \rangle$, denoted by $I \models Q$, if for any relation $r \in \mathcal{R}$, $L(r) \subseteq$ supp $I(r) \subseteq U(r)$, and ϕ holds. Additionally, the solver will try to optimize the distance of the newly generated solution from the set $\mathcal{S}$. We will allow $\mathcal{S}$ to contain both qualitative and quantitative bindings, and assume that the model finder adapts the distance measures accordingly. (More about this in Section 4.)

As a simple example, consider a problem with 3 relations declared with the following lower- and upper-bounds: unary (i.e., sets) $\{\} \subseteq X \subseteq \{(x), (y)\}$ and $\{\} \subseteq Y \subseteq \{(w), (z)\}$, and binary $\{\} \subseteq R \subseteq \{(x, w), (x, z), (y, w), (y, z)\}$, and a constraint ϕ imposing $R \subseteq X \times Y$. A possible valid solution for $\mathbb{D} = \mathbb{Z}$ would be

the following, omitting the zero-valued tuples:

$$I = \{X \mapsto (x) \mapsto 1, Y \mapsto (w) \mapsto 1, Y \mapsto (z) \mapsto 2,$$
$$R \mapsto (x, w) \mapsto 5, R \mapsto (x, z) \mapsto 7\} \quad .$$

An *iteration* is an operation that, given a problem Q and a solution (quantitative binding) I, returns a new problem to generate a new solution. For instance, the naive iteration, denoted by next, is simply defined as:

$$\mathsf{next}(\langle \mathcal{M}, L, U, \phi, \_ \rangle, I) = \langle \mathcal{M}, L, U, \phi \wedge \neg [\![I]\!], \emptyset \rangle$$

where $[\![I]\!]$ produces a formula that exactly represents I, which is excluded as a possible solution by being negated into ϕ. $\mathcal{S}$ is set to $\emptyset$ since there is no distance goal in next. This guarantees that no identical instance is returned, i.e.:

$$\forall I' \cdot I' \models \mathsf{next}(Q, I) \Rightarrow I \neq I'$$

Subsequent applications of next accumulate negated solutions in ϕ, so no two identical instances are ever returned.

Structure-level iteration These iterations, which we shall denote as variants of a next_S operation for different optimization goals, are expected to produce a new solution with a different structure, i.e., abiding to the following property:

$$\forall I' \cdot I' \models \mathsf{next}_S(Q, I) \Rightarrow \text{supp } I \neq \text{supp } I'$$

Without a distance goal, this operation essentially removes the current structure from the search space:

$$\mathsf{next}_S^{\emptyset}(\langle \mathcal{M}, \_, \_, \phi, \_ \rangle, I) = \langle \mathcal{M}, L_0, U_0, \phi \wedge \neg [\![\text{supp } I]\!], \emptyset \rangle$$

Note the abuse of notation in denoting by $[\![\text{supp } I]\!]$ the translation of a qualitative binding into a formula that just determines whether tuples are present in the solution, regardless of the associated quantity (i.e., whether their value is zero). Notice how the bounds must be reset by this operation: the quantitative-level iterations (presented next), force a particular structure by tightening the bounds, which must be reset whenever a new structure is requested.

Without any optimization goal, $\mathsf{next}_S^{\emptyset}$ behaves essentially like a qualitative iteration, ignoring the quantities. If we wish to explore more varied structures, a distance goal must be set. This is similar to target-oriented operations proposed for the qualitative context [11]. As an example, consider an operation next_S^B that tries to produce maximally different structures.

$$\mathsf{next}_S^B(\langle \mathcal{M}, \_, \_, \phi, \_ \rangle, I) = \langle \mathcal{M}, L_0, U_0, \phi \wedge \neg [\![\text{supp } I]\!], \{\text{supp } I\} \rangle$$

The distance goal is simply the support of the previous instance, so that the structure of the next solution is as distinct as possible. Notice that the previous goal is ignored, including any instances accumulated by the quantitative-level iterations (presented next).

Quantity-level iteration Enumerating over quantities of a particular structure, a class of operations to be denoted by variants of next_Q, must satisfy two invariants: that the solutions preserve the specified structure throughout such enumeration steps (or until there are no more solutions of this kind respecting the model's constraints), and that some quantity changes:

$$\forall I' \cdot I' \models \mathsf{next}_Q(Q, I) \Rightarrow \text{supp } I = \text{supp } I' \wedge I \neq I'$$

The naive quantitative-level iteration $\mathsf{next}_Q^{\emptyset}$ will allow quantities to change arbitrarily. Formally:

$$\mathsf{next}_Q^{\emptyset}(\langle \mathcal{M}, \_, \_, \phi, \_ \rangle, I) = \langle \mathcal{M}, \text{supp } I, \text{supp } I, \phi \wedge \neg [\![I]\!], \emptyset \rangle$$

Note how the lower- and upper-bounds are fixed with the support of I, forcing the exact same tuples to exist in the next solution.

To produce solutions that are maximally distinct, note that one cannot simply ask for quantities further away from the current I, because in dense domains such as the fuzzy one, this could result in a "ping-pong" behaviour between two sets of similar solutions. Thus, we must accumulate all previously seen solutions as the distance goal. Structure-level iterations always reset $\mathcal{S}$, so these operations only consider the distance to solutions for the currently fixed structure.

$$\mathsf{next}_Q^{\mathcal{S}}(\langle \mathcal{M}, \_, \_, \phi, \mathcal{S} \rangle, I) = \langle \mathcal{M}, \text{supp } I, \text{supp } I, \phi \wedge \neg [\![I]\!], \mathcal{S} \cup \{I\} \rangle$$

4 Quantitative Iteration in QAlloy

The authors have previously proposed an extension to Alloy to allow quantitative relational modelling – QAlloy — which currently supports both the integer and fuzzy domains. This section describes how the iteration operations from the previous section were integrated into the model finding backend of QAlloy, briefly introducing the QAlloy infrastructure as needed.[8] For a more detailed description of QAlloy, the interested reader is redirected to [22,21].

QAlloy and its Analyzer At the language level, the key extension of QAlloy is the introduction of n-ary quantitative relations through special keywords, whose tuples will be assigned values from the selected domain ($\mathbb{Z}$ for **`int`** and $[0, 1] \subseteq \mathbb{R}$ for **`fuzzy`**), the 0 quantity representing the tuple's absence from the relation. The semantics of the relational operators (e.g. relational composition) were adapted accordingly, preserving the bulk of the language and retro-compatibility in case only Boolean relations are used. A particularity is that numerical constants cannot be declared standalone, but rather associated with an element from $\mathcal{U}$, i.e., they are "typed" and thus benefit from the Alloy's type system. This promotes a more rigorous usage of *units*, which are often the source of issues in software (e.g., combining two quantities of "incompatible units" throws a type error).

[8] QAlloy is an extension to Alloy 5; migration into Alloy 6 is underway, but currently the temporal aspect introduced in this latest release is not supported.

Both the Alloy Analyzer and the underlying Kodkod [27] relational model finder have been extended to handle quantitative relational models. Kodkod's Boolean structures managing relations and operations between them were extended to numerical ones, namely representing quantitative relations through numerical matrices. Relational operations are thus defined by linear algebra. To actually solve a model finding problem, Kodkod's SAT solver backend was switched to one that relies on off-the-shelf SMT solvers. The results are then provided back to the Analyzer whose visualizer and evaluator were adapted to the quantitative context, as shown in the examples of Section 2.

The two new classes of iterations, whose implementation is detailed next, are provided to the user in the visualizer's toolbar with buttons "`Next Structure`" and "`Next Quantity`", in contrast to the single "`Next`" of Alloy 5.

Quantitative iteration in Kodkod As highlighted in Section 2, Kodkod's iteration (operation `next`) is not effective in quantitative problems. Thus, the alternative iteration methods proposed in Section 3 were implemented at the Kodkod level. Since these require optimization capabilities, we rely on Max-SMT solvers [16] which allow the definition of optimization goals.

The SMT specification for a given quantitative model finding problem Q defines its primary variables through *function symbols*. When $\mathbb{D} = \mathbb{Z}$, the function symbols are declared as integer-valued, with the solving process being executed according to the *Theory of Integers* using the logical fragment `QF_NIA`; for $\mathbb{D} = [0,1] \subseteq \mathbb{R}$, the function symbols are specified as reals and the `QF_NRA` of the *Theory of Reals* is used instead. Each free n-ary relation $r \in \mathcal{R}$ is encoded as a $|\mathcal{U}|^n$ matrix; each matrix element gives origin to a free variable at SMT-level, denoting the value of an n-ary tuple t in r. Let `r_t` be such a primary SMT variable. If a tuple t is in the lower-bound of $L(r)$, `r_t` is forced is be non-zero; if it is outside the upper-bound of $U(r)$, it is forced to be zero.

Table 1. SMT encoding of bindings for n_i-ary relations $r_i \in \mathcal{R}$ and tuples $t_j^{n_i} \in \mathcal{T}_\mathcal{U}$

Binding	**SMT encoding**	$f(r,t)$
$\neg[\![B]\!]$	`(assert (or` $f(r_0,t_0^{n_0})$ $f(r_0,t_1^{n_0})$ $\ldots$ $f(r_k,t_{\lvert\mathcal{U}\rvert^{n_k}}^{n_k})$`))`	`(not (= r_t 0))` if $B(r)(t)=0$ `(= r_t 0)` otherwise
$\neg[\![I]\!]$	`(assert (or` $f(r_0,t_0^{n_0})$ $f(r_0,t_1^{n_0})$ $\ldots$ $f(r_k,t_{\lvert\mathcal{U}\rvert^{n_k}}^{n_k})$`))`	`(not (= r_t v))` where $v = I(r)(t)$

Constraints over function symbols are imposed through *assertions*, which are managed in the solver's **Assertion Stack**. This structure supports *push* and *pop* operations to manage assertions at different levels, and commands such as e.g. `(check-sat)` — which is used to determine the satisfiability of the model — take into account every declaration and assertion up to the level of the stack where it is being called. Thus, although formally a new problem is generated whenever an iteration is applied, for performance issues we actually preserve the SMT problem running between iterations and update the stack accordingly. The problem's constraint ϕ is set at the base level of the assertion stack, meaning that

it is always considered; meanwhile, temporary constraints might be considered in a new level through *push*, and later on may be disregarded using *pop*, enabling the application of different iteration operations. Two kinds of additional constraints can occur in a problem Q: (a) imposing additional lower- or upper-bounds adds constraints to the primary variables, which are popped when reset into L_0 and U_0; and (b) introducing additional constraints $\neg[\![B]\!]$ and $\neg[\![I]\!]$ for qualitative and quantitative bindings, respectively, whose translation is shown in Table 1. These assertions are pushed after every solution is found and are never popped, so the removal of a solution from the search space is permanent.

Table 2. SMT incremental goals for n_i-ary relations $r_i \in \mathcal{R}$ and tuples $t_j^{n_i} \in \mathcal{T}_\mathcal{U}$

Goal	**SMT encoding**	$d(r,t)$

B	`(assert-soft` $d(r_0, t_0^{n_0})$`)`
	`(assert-soft` $d(r_0, t_1^{n_0})$`)`
	…
	`(assert-soft` $d(r_k, t_{\lvert\mathcal{U}\rvert^{n_k}}^{n_k})$`)`
		$\begin{cases} \texttt{(not (= r\_t 0))} & \text{if } B(r)(t) = 0 \\ \texttt{(= r\_t 0)} & \text{otherwise} \end{cases}$

I_{SDST}	`(maximize (+` $d(r_0, t_0^{n_0})$ $d(r_0, t_1^{n_0})$ … $d(r_k, t_{\lvert\mathcal{U}\rvert^{n_k}}^{n_k})$`)))`	

I_{VDST}	`(maximize` $d(r_0, t_0)$`)`
	`(maximize` $d(r_0, t_1)$`)`
	…
	`(maximize` $d(r_k, t_{\lvert\mathcal{U}\rvert^{\lvert r_k\rvert}})$`)`
		`(ite (> r_t v) (- r_t v) (- v r_t))`
		where $v = I(r)(t)$

Max-SMT solvers provide optimization commands such as `(maximize t)`, which signal the solver to maximize the value of term `t` according to the stack's assertions; and `(assert-soft a)` to declare soft-constraints, whose validity is not required, but the solver will attempt to maximize the number of soft-constraints which can hold. These features can be used to implement different goals in guided iterations. The goal is to generate a solution that is as further away from a set of bindings $\mathcal{S}$, but since $\mathcal{S}$ grows incrementally with each new instance, we can also incrementally accumulate goals in the assertion stack (until it is reset by a structure-level operation, in which case the goals must be popped). Table 2 shows the encoding for single qualitative and quantitative bindings. It should be noted that the shown encoding implements the Manhattan distance between bindings (i.e., the sum of differences), rather than the more classical Euclidean distance. Our early experiments showed that the latter weighted heavily on the underlying solvers, due to the usage of exponentiation and square root operations. Evaluation in the next section will show that this still produced reasonable results with acceptable performance.

Two alternative encodings are shown for a quantitative binding I: one maximizes the distance to the instance (SDST), the other the distance to each tuple of the instance individually (VDST). Although these are semantically equivalent, Max-SMT solvers perform differently in multi-objective optimization problems. In order to further mitigate this issue, we explore alternative goal implementations that reduce the number of objectives by considering the average of all seen bindings (SAVG and VAVG, respectively). The trade-off is that these averages

Table 3. SMT batch goals for n_i-ary relations $r_i \in \mathcal{R}$ and tuples $t_j^{n_i} \in \mathcal{T}_\mathcal{U}$

Goal	**SMT encoding**	$d(r,t)$
I_{SAvG}	`(maximize (+` $d(r_0, t_0^{n_0})$ $d(r_0, t_1^{n_0})$ `...` $d(r_k, t_{\lvert\mathcal{U}\rvert^{n_k}}^{n_k})$`))`	

I_{VAvG}	`(maximize` $d(r_0, t_0^{n_0})$`)`
	`(maximize` $d(r_0, t_1^{n_0})$`)`
	...
	`(maximize` $d(r_k, t_{\lvert\mathcal{U}\rvert^{n_k}}^{n_k})$`)`
		`(ite (> r_t v) (- r_t v) (- v r_t))`
		where $v = \frac{\sum_{I \in \mathcal{S}} I(r)(t)}{\lvert\mathcal{S}\rvert}$

are updated in each iteration, so these objectives must be popped from the stack at each iteration. Their encoding is shown in Table 3 (focusing on quantitative cases, since structure-level goals consider a single previous binding). Max-SMT solvers use different **priorities** towards which the goals should be optimized, including a *lexicografic* order; an *independent box objective* that attempts to find a solution where every goal is maximal; or through *pareto fronts*. These vary in performance and may reach different solutions; we evaluate these alternatives in the next section.

Unbounded domains As one might expect, when using optimization goals for problems which are neither implicitly nor explicitly bounded, the optimization goal may be infinitely valued. In these situations Max-SMT solvers report that a goal is unbounded, alongside a non-optimal solution. Our current implementation reports this to the user, but still providing the valid, but not-optimal, solution. The user is nonetheless advised to restrict the quantitative model with additional constraints to bound the values of the quantitative relations.

5 Evaluation

This section evaluates the proposed quantitative solution-iterations and their implementation in a Max-SMT backend through QAlloy. We shall be interested in assessing not only performance but also how varied the generated solutions are. While we are aware that assessing the quality of the generated instances requires proper user studies, we consider their variety to be a prerequisite for useful iteration sessions, this leading to the following research questions:

RQ1 How efficient is the SMT implementation of the quantitative iterations?
RQ2 Which iterations produce more varied solutions within their class?
RQ3 How efficient is the detection of infinitely-valued optimization goals?

To answer these questions, we consider several QAlloy examples. We consider a very simple model of a point in the Cartesian plane in both the integer and fuzzy domains, which proved useful to visualize the impact of the iterations in the solution space. In the integer domain, these include the supermarket self-checkout system addressed in Section 2; a model of quantitative properties over vertex-labelled graphs; a model for a simple electronic purse and the transference operations between them; two variants of flow networks, with the flow that

passes through the network being untyped in the first example, and explicitly typed as *litres* in the second. In the fuzzy domain, problems include the QAlloy medical diagnosis showcased in Section 2, as well as an *intuitionistic* variant [5]; fuzzy clustering applied to grouping portraits based on visual similarity [25]; two distinct models of fuzzy controllers, one an automatic heater using the Mamdani fuzzy inference system, and the other a tipping system according to the service and food appraisal, modelled as a Sugeno fuzzy inference system [12,24,18].

We selected 19 commands for these models, all satisfiable in order to allow for iteration (i.e., either run commands generating instances or check commands generating counterexamples).[9] To answer **RQ1** and **RQ2**, all these commands acted on variables bounded by the model. To answer **RQ3**, variants of 4 commands were considered that left some variables unbounded. The ν_Z [2] (v4.8.12) Max-SMT solver was used at the backend.[10] All commands were run with the alternative distance measures presented in Section 4, and with alternative priority strategies in multi-objective problems. All tests were run in a machine equipped with a deca-core Intel Core i7-1355U @ 1.30 GHz with 16GB of RAM. QAlloy ran with 8192MB maximum memory and 8192k of maximum stack size. For all commands, each iteration operation was applied up to 30 times or until unsatisfiable (i.e., no more solutions). For the initial execution of a command, a timeout of 10 minutes was considered, while a timeout of 1 minute was set for every iteration step. All QAlloy models, the benchmark script and the execution results, as well as summary tables, are publicly available [20].

Table 4. Aggregated results for the first 10 applications of next_S variants.

Operation	$\mu_{min}(ms)$	$\mu_{max}(ms)$	μ_N	σ_N	$t_{out}(\%)$	$\mu_{<1}(\%)$	$\mu_{<10}(\%)$	μ_D	σ_D
next	2	1928	4	24	0.0	94.74	100.0	5	67
$\mathsf{next}_S^{\emptyset}$	1	848	13	47	5.26	94.74	94.74	66	22
next_S^{B}	3	5812	100	65	10.53	68.42	89.47	100	57

Tables 4 and 5 summarize the results for the first 10 instances produced with structure- and quantity-level iterations, respectively. Each G_P in column "Goal" of Table 5 represents running operation I_G (see Tables 2 and 3), employing the priority P for those implementing multiple optimization goals (either "LEXicographic order", "independent BOX objective" or "PAReto fronts"). Regarding execution time, μ_{min} and μ_{max} show the lowest and highest (excluding timeouts) response times produced by each method, in milliseconds; columns μ_N and σ_N display the response times after normalization between commands (ranging from 0 to 100, the lower the better); t_{out} measures the % of commands which reached a timeout, and $\mu_{<1}$ and $\mu_{<10}$ show the % of commands whose response time was lower than 1 and 10 seconds on average, respectively. Re-

[9] For fuzzy problems, the popular Gödelian t-norm was selected.

[10] We focused on ν_Z as the most popular and stable Max-SMT solver, but it could be easily swapped by others conforming to SMT-LIB.

Table 5. Aggregated results for the first 10 applications of next_Q variants.

Operation	Goal	$\mu_{min}(ms)$	$\mu_{max}(ms)$	μ_N	σ_N	$t_{out}(\%)$	$\mu_{<1}(\%)$	$\mu_{<10}(\%)$	μ_D	σ_D
$\mathsf{next}_Q^{\emptyset}$	NA	2	187	0	0	0.0	100.0	100.0	11	15
$\mathsf{next}_Q^{\mathcal{S}}$	SDst$_{\text{BOX}}$	18	3934	74	74	21.05	47.37	78.95	17	45
	SDst$_{\text{LEX}}$	14	4352	33	23	15.79	78.95	84.21	12	42
	SDst$_{\text{PAR}}$	7	34	13	4	57.89	42.11	42.11	21	26
	VDst$_{\text{BOX}}$	22	6773	73	77	31.58	52.63	68.42	26	24
	VDst$_{\text{LEX}}$	13	5969	30	17	10.53	78.95	89.47	9	37
	VDst$_{\text{PAR}}$	15	33	16	4	68.42	31.58	31.58	13	17
	VAvg$_{\text{BOX}}$	8	11512	25	13	21.05	73.68	73.68	51	31
	VAvg$_{\text{LEX}}$	7	1706	16	2	10.53	84.21	89.47	74	69
	VAvg$_{\text{PAR}}$	9	37	43	6	68.42	31.58	31.58	33	29
	SAvg	5	7082	26	6	10.53	84.21	89.47	76	63

garding the variety of the produced solutions, μ_D and σ_D evaluate the average distance between every pair of generated solutions, also normalized for each command. For structure-level iterations, quantities are disregarded when measuring distances, amounting to the set difference of the supports. For quantity-level iterations, although the implementation relied on the Manhattan distance, for evaluation we consider the ideal Euclidean distance. In order for the comparison to be fair, only commands that do not hit a timeout are considered in these metrics (since a smaller number of instances is more likely to be varied). For example, VDst$_{\text{PAR}}$, shows low response times, but a high timeout rate of $\sim 70\%$; the times measured refer only to the $\sim 30\%$ commands where it did *not* timeout. Table 4 also presents the results for the naive approach as a baseline, but this comparison must be made with care since next was not designed to produce varied solutions.

RQ1 and RQ2 Focusing first on the performance of structure-level iterations (**RQ1**), $\mathsf{next}_S^{\emptyset}$ finds solutions in line with the naive approach next (its response time, bar timeouts, does not exceed $0.9s$ (seconds) on average), while next_S^{B} is an order of magnitude slower but without exceeding $6s$ on average. Moreover, both methods are overall consistent in finding answers without hitting timeout, with a low rate of $\sim 5\%$ and $\sim 10\%$, respectively. Regarding **RQ2**, the difference in variety from the baseline next to both methods is large, as expected; moreover, next_S^{B} consistently produces the most varied set of solutions for almost every command. When extending the analysis to the first 30 solutions (full data available at [20]), there is a slight increase of timeouts for the naive approach and $\mathsf{next}_S^{\emptyset}$, but not for next_S^{B}, without significant changes in the normalized variety of solutions.

Moving on to quantity-level iterations, we consider $\mathsf{next}_Q^{\emptyset}$ as a baseline, since it fixes the structure of the solution but changes quantities arbitrarily (next can change the structure, making the comparison more complicated). Regarding response time (**RQ1**), it is apparent that in multi-objective encodings the pareto (PAR) priority results in timeouts for most commands ($57 \sim 69\%$). At the other end of the spectrum, VDst$_{\text{LEX}}$, VAvg$_{\text{LEX}}$ and SAvg display the lowest

amounts of timeouts ($\sim 10\%$), while VAvg$_{\text{LEX}}$, SAvg and VAvg$_{\text{BOX}}$ have the best performance (lowest average mean and average standard deviation); when these methods do not timeout, they do not exceed $12s$ on average. Focusing on the distance metrics (**RQ2**), the same configurations overall show significant improvements in the solution variety. VAvg$_{\text{BOX}}$ (high mean and low variance), VAvg$_{\text{LEX}}$ (high mean and highest variance) and SAvg (highest mean and high variance) are the quantity-level iterations generating solutions in a richer way. Note that higher values of mean and standard deviation indicate larger differences between the solutions produced. When considering the first 30 solutions, there is a noticeable decrease in solution distance (which is expected, since more of the search space has been covered), but VAvg$_{\text{BOX}}$, VAvg$_{\text{LEX}}$ and SAvg still show the best variety. SDst$_{\text{BOX}}$ and VDst$_{\text{BOX}}$ suffer from an increase in timeouts (of $\sim 20\%$ and $\sim 10\%$, respectively) and their average response times are on the higher end. This is perhaps not surprising, as both methods cumulatively add assertions to the stack in each iteration, so while the initial solutions may be found faster and be varied, it will be harder for the solver to find varied solutions while meeting every optimization goal. Besides these, SDst$_{\text{LEX}}$ shows an increase of $\sim 5\%$ timeouts, while the rate for the remaining quantity-level methods is unaffected.

Table 6. Examples which lead to infinite optimization goals.

Operation	Goal	$\mu_{min}(ms)$	$\mu_{max}(ms)$	μ_N	σ_N	$i_{tout}(\%)$	$t_{out}(\%)$	$\mu_{<1}(\%)$	$\mu_{<10}(\%)$	μ_D	σ_D
$\mathsf{next}_Q^{\emptyset}$		2	29	0	6	0	0	100	100	0	2
next_Q^{S}	SDst$_{\text{BOX}}$	7	5975	76	75	0	0	75	100	54	52
	SDst$_{\text{LEX}}$	4	119	14	14	0	0	100	100	10	8
	SDst$_{\text{PAR}}$	*NA*	*NA*	*NA*	*NA*	0	100	0	0	*NA*	*NA*
	VDst$_{\text{BOX}}$	6	947	87	90	25	50	50	50	50	50
	VDst$_{\text{LEX}}$	5	60	21	9	0	0	100	100	5	2
	VDst$_{\text{PAR}}$	*NA*	*NA*	*NA*	*NA*	100	100	0	0	*NA*	*NA*
	VAvg$_{\text{BOX}}$	4	29	13	0	0	50	50	50	59	51
	VAvg$_{\text{LEX}}$	3	46	7	6	0	0	100	100	33	28
	VAvg$_{\text{PAR}}$	*NA*	*NA*	*NA*	*NA*	100	100	0	0	*NA*	*NA*
	SAvg	3	48	7	0	0	0	100	100	31	26

RQ3 Table 6 addresses iterations that make use of Max-SMT optimization capabilities, for commands that lead to unbounded optimization goals, resulting in an infinite outcome (a special case of a satisfiable outcome). It has an extra column i_{tout} presenting the % of commands that reached a timeout on the *first* iteration, representing cases where the user is not informed of the unbounded nature of the command; t_{out} still identifies the cases where a timeout was *eventually* reached. Column i_{tout} shows three cases where the user is not properly informed. Notice also that there are now no methods on the higher-end of the distance metric; since solutions are no longer optimal, results are much less predictable.

6 Related Work

There is considerable literature on controlling how solutions are generated in *qualitative* model finding. The authors of [11] introduce the concept of weighted target-oriented model finding, which was subsequently used in solution enumeration operations [11]. These relied on (partial) maximum satisfiability (Max-SAT) solvers, as do subsequent approaches, namely Aluminum [15] and Bordeaux [13]. REACH [10] enumerates solutions ordered by size (smallest to largest). At the GUI-level, HawkEye [23] allows the user to control the next solutions by manipulating the instance in the visualizer. Few works have tried to effectively measure the impact of the different provided solutions through user studies [4,3].

Focusing on SMT solvers, path exploration work (e.g. [28]), inspired by testing techniques, tries to generate instances that cover different valuations of SMT formulas; SMT sampling approaches focus on generating stimuli to ensure "good coverage" (e.g. according to user-defined criteria) of the solution space. They include SMTSampler [7] and GuidedSampler [8], which only support theories of bit-vectors, arrays, and uninterpreted functions, and MeGASampler [17], that provides an heuristic using a theory of intervals to support the theory of integer arithmetic (without division). However, it remains unclear how iteration at the SMT-level would reflect at more abstract level of model finding problems. In contrast, similarly to qualitative approaches based on Max-SAT, we took advantage of Max-SMT solvers to achieve diverse quantitative enumeration, whose distance goals are provided at the model finding problem level. Moreover, many of these approaches require further user input, while we believe that, at model finding level operations should be provided with "push-button" simplicity.

Another technique employed by model finders is symmetry breaking, to avoid the generation of isomorphic solutions, which Kodkod in particular implements at the SAT level [27]. There is some work on symmetry breaking at the SMT level. SyMT [1] provides an approach centered in building coloured graphs from SMT formulas and finding said graphs' automorphisms, representing the problem's symmetries. CVC4-SymBreak [6] makes use of the SyMT tool, but rather to exploit the symmetries of the SAT module of the SMT solving process.

7 Conclusions and Future Work

This paper proposes a formalization of solution enumeration operations in quantitative relational model finding, considering distinct operations for changes to the structure and to the quantities. An implementation of these operations on top of Max-SMT solvers is provided, whose evaluation shows to be performant and capable of generating solutions with a high degree of variability.

There are a few directions along which this work may evolve. Symmetry breaking is still unaddressed at the quantitative model finding level, despite works at the SMT level. Our formalization could be extended with further information, such as providing different weights to the relations, or considering user information provided through the instance visualizer. More advanced operations

may require information from the problem's constraints (e.g., to satisfy different properties regardless of the distance to the seen solutions). Relevant insights may come from data-driven search-based techniques [14] used in software engineering, for instance, for test case generation.

QAlloy is also being migrated to the most recent Alloy 6 release, whose temporal dimension introduces an additional layer of complexity in iteration.

Last but not least, and perhaps most important: to effectively assess the quality of the generated solutions, thorough user studies must be performed.

Acknowledgements

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020 (DOI 10.54499/LA/P/0063/2020). J.N. Oliveira is also supported by project FCT: PTDC/CCI-COM/4280/2021 and P. Silva holds FCT grant 2023.01186.BD.

References

1. Areces, C., Déharbe, D., Fontaine, P., Ezequiel, O.: SyMT: Finding symmetries in SMT formulas. In: SMT Workshop (2013)
2. Bjørner, N.S., Phan, A.: νz-maximal satisfaction with Z3. In: SCSS. EPiC Series in Computing, vol. 30, pp. 1–9. EasyChair (2014)
3. Cunha, A., Macedo, N., Campos, J.C., Margolis, I., Sousa, E.: Assessing the impact of hints in learning formal specification. In: SEET@ICSE. pp. 151–161. ACM (2024)
4. Danas, N., Nelson, T., Harrison, L., Krishnamurthi, S., Dougherty, D.J.: User studies of principled model finder output. In: SEFM. LNCS, vol. 10469, pp. 168–184. Springer (2017)
5. De, S.K., Biswas, R., Roy, A.R.: An application of intuitionistic fuzzy sets in medical diagnosis. Fuzzy Sets Syst. **117**(2), 209–213 (2001)
6. Dingliwal, S., Agarwal, R., Mittal, H., Singla, P.: Advances in symmetry breaking for SAT Modulo Theories (2020), `https://arxiv.org/abs/1908.00860`
7. Dutra, R., Bachrach, J., Sen, K.: SMTSampler: Efficient stimulus generation from complex SMT constraints. In: ICCAD. p. 30. ACM (2018)
8. Dutra, R., Bachrach, J., Sen, K.: GUIDEDSAMPLER: Coverage-guided sampling of SMT solutions. In: FMCAD. pp. 203–211 (2019)
9. Jackson, D.: Alloy: A language and tool for exploring software designs. Communications of the ACM **62**(9), 66–76 (2019)
10. Jovanovic, A., Sullivan, A.: REACH: Refining Alloy scenarios by size (tools and artifact track). In: ISSRE. pp. 229–238. IEEE (2022)
11. Macedo, N., Cunha, A., Guimarães, T.: Exploring scenario exploration. In: FASE. LNCS, vol. 9033, pp. 301–315. Springer (2015)
12. Mamdani, E., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. Journal of Man-Machine Studies **7**(1), 1–13 (1975)
13. Montaghami, V., Rayside, D.: Bordeaux: A tool for thinking outside the box. In: FASE. LNCS, vol. 10202, pp. 22–39. Springer (2017)
14. Nair, V., Agrawal, A., Chen, J., Fu, W., Mathew, G., Menzies, T., Minku, L., Wagner, M., Yu, Z.: Data-driven search-based software engineering. CoRR **abs/1801.10241** (2018)

15. Nelson, T., Saghafi, S., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Aluminum: Principled scenario exploration through minimality. In: ICSE. pp. 232–241. IEEE Computer Society (2013)
16. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and optimization problems. In: SAT. LNCS, vol. 4121, pp. 156–169. Springer (2006)
17. Peled, M., Rothenberg, B., Itzhaky, S.: SMT sampling via model-guided approximation. In: FM. LNCS, vol. 14000, pp. 74–91. Springer (2023)
18. Reznik, L.: Fuzzy controllers handbook: How to design them, how they work. Elsevier (1997)
19. Sanchez, E.: Solutions in composite fuzzy relation equations: Application to medical diagnosis in brouwerian logic. In: Readings in Fuzzy Sets for Intelligent Systems, pp. 159–165. Morgan Kaufmann (1993)
20. Silva, P.: QAlloy repository – quantitative solution iteration (2025), `https://github.com/pf7/QAlloy-QSI`
21. Silva, P., Cunha, A., Macedo, N., Oliveira, J.N.: Alloy goes fuzzy. In: ABZ. LNCS, vol. 14759, pp. 61–79. Springer (2024)
22. Silva, P., Oliveira, J.N., Macedo, N., Cunha, A.: Quantitative relational modelling with QAlloy. In: ESEC/SIGSOFT FSE. pp. 885–896. ACM (2022)
23. Sullivan, A.: HawkEye: User-guided enumeration of scenarios. In: ISSRE. pp. 569–578. IEEE (2021)
24. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE Trans. Syst. Man Cybern. **15**(1), 116–132 (1985)
25. Tamura, S., Higuchi, S., Tanaka, K.: Pattern classification based on fuzzy relations. IEEE Trans. Syst. Man Cybern. **1**(1), 61–66 (1971)
26. Tarski, A., Givant, S.: A Formalization of Set Theory without Variables. AMS (1987), AMS Colloquium Publications, volume 41.
27. Torlak, E., Jackson, D.: Kodkod: A relational model finder. In: TACAS. LNCS, vol. 4424, pp. 632–647. Springer (2007)
28. Wu, H.: Generating metamodel instances satisfying coverage criteria via SMT solving. In: MODELSWARD. pp. 40–51. SciTePress (2016)
29. Zhang, C., Wagner, R., Orvalho, P., Garlan, D., Manquinho, V., Martins, R., Kang, E.: AlloyMax: Bringing maximum satisfaction to relational specifications. In: ESEC/SIGSOFT FSE. pp. 155–167. ACM (2021)