Comparing the Effectiveness of Reasoning Formalisms for Partial Models

Pooya Saadatpanah Michalis Famelis Jan Gorzny Nathan Robinson
Marsha Chechik Rick Salay

University of Toronto, Toronto, Canada
{pooya, famelis, jgorzny, nrobinso, chechik, rsalay}@cs.toronto.edu

ABSTRACT

Uncertainty is pervasive in Model-based Software Engineering. In previous work, we have proposed *partial models* as a way to explicate uncertainty during modeling. Using partial models, modelers can perform certain forms of reasoning, like checking properties, without the having to prematurely resolve uncertainty. In this paper, we present a strategy for encoding partial models into different reasoning formalisms and conduct an empirical study aimed to compare the effectiveness of these formalisms for checking properties of partial models.

1. INTRODUCTION

Modelers are routinely called upon to work with artifacts that contain varying degrees of uncertainty. However, existing modeling methodologies, tools, and libraries usually assume unambiguous artifacts. Modelers are thus often forced to act as if they were certain, artificially removing uncertainty from their artifacts. This carries the risk of making premature decisions, which can have significant effects on the quality of the produced software.

In [5], we introduced the idea of using *partial models* as first class artifacts to explicitly handle uncertainty by allowing the deferral of uncertainty resolution in model-based software development. A partial model is a model that represents a set of conventional models. In subsequent work, we've studied various aspects of this approach, such as property checking and diagnosis [6], refinement [15] and transformation [7]. In [17], we introduced *MAVO partiality*, as a way to explicate uncertainty using syntactic annotations. We introduced four kinds of such annotations:

- ***M**ay partiality*: annotating a model element with M indicates that we are unsure about whether it should exist in the model or not,
- ***A**bs partiality*: annotating an element with S indicates that we are unsure about whether it should actually be a collection of elements,
- ***V**ar partiality*: annotating an element with V indicates that we are unsure about whether it should actually be merged with other elements, and
- ***O**W partiality*: annotating the entire model with INC indicates that we are unsure about whether it is complete.

Figure 1(a) shows a simple partial class diagram M_1. In addition to the class `Vehicle`, M_1 has several points of uncertainty, explicated with MAVO annotations: (a) We are uncertain whether we need to create separate classes for `LandVehicle`s. (b) If we do that, we do not know how many such classes we may need to create. (c) We are not sure whether they should be subclasses of `Vehicle`. (d) We do not know where we should add the attribute `numDoors` in this hierarchy. Figure 1(b) shows a conventional model m_1 where all these points of uncertainty have been resolved. Such models that can result from systematically removing uncertainty from a MAVO model are called *concretizations*. In this example, `LandVehicle` has been refined to three classes `Car`, `Truck` and `Motorcycle`. The attribute `numDoors` has been assigned to the class `Vehicle` and thus the class `Motorcycle` does not inherit from `Vehicle`.

An important benefit of partial models is that we can check their properties without the need to remove uncertainty, thus facilitating decision deferral. For example, we might be interested in checking whether there can exist a refinement of M_1 that has cycles in the inheritance hierarchy.

In [17], we gave semantics to the MAVO annotations using First Order Logic (FOL) and demonstrated how they can be used for property checking using Alloy [11]. In this paper, we compare Alloy with three other reasoning formalisms, CSP [19], SMT [3] and ASP [12], aiming to determine which is most efficient for checking properties of partial models. We focus exclusively on the first three partiality types (*May*, *Abs*, *Var*). In the future, we intend to also study reasoning with *OW*, which is significantly harder, as it relaxes the closed world assumption. To facilitate meaningful comparison, we created an encoding of MAVO in Relational Algebra that can be readily translated into the different modeling formalisms. Using this encoding, we set up a series of experiments using randomly generated MAVO models.

The rest of the paper is structured as follows: Section 2 gives necessary background on MAVO models and the reasoning formalisms. Section 3 introduces the relational encoding of MAVO. Section 4 describes the experiments for comparing the reasoning formalisms. We discuss related work in Section 5 and conclude in Section 6.

2. BACKGROUND

MAVO models. In this paper, we consider models expressed as typed, directed graphs. The model M_1 in Figure 1(a) is shown in

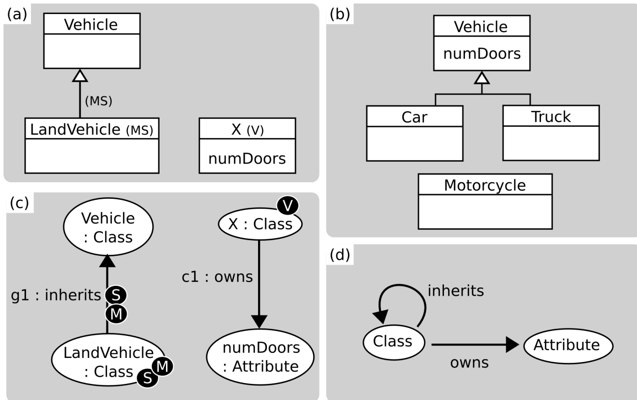

Figure 1: (a) Example MAVO model M_1. (b) A concretization m_1 of M_1. (c) M_1 expressed as an annotated, typed graph G_{M_1}. (d) G_{M_1}'s type graph G_{CD}.

its abstract syntax as a typed graph G_{M_1} in Figure 1(c). Classes and attributes are represented in G_{M_1} as typed nodes; inheritance and attribute ownership as typed edges. A typed graph's nodes and edges are called its *elements*. G_{M_1} conforms to the *type graph* G_{CD}, shown in Figure 1(d), that corresponds to the signature of a very simple UML Class diagram metamodel. G_{CD} specifies that there exist `Class`es and `Attribute`s, that one `Class` can `inherit` another, and that `Class`es can `own` `Attribute`s.

A MAVO model is simply a typed graph whose elements can be decorated with the MAVO annotations. Semantically, a MAVO model represents a set of conventional models (its *concretizations*). We consider a non-annotated model to be a MAVO model that represents a set containing exactly one concretization. In Figure 1(c), MAVO annotations are shown in black circles. For example, the attribute `numDoors` is owned by the class `X` annotated with V, indicating that it can be merged with some other `Class` node.

In Section 1, we introduced the idea that a MAVO model can be refined [17] to conventional models, based on the MAVO annotations of its elements. This is done by refining the individual elements of the MAVO model ("*MAVO elements*") to elements in the concretization ("*instance elements*"), according to the definitions of the different MAVO annotations. For example, in the concretization m_1, shown in Figure 1(b), the instance elements *Car*, *Truck* and *Motorcycle* refine the S-annotated MAVO element `LandVehicle` of M_1.

Reasoning formalisms. In this section, we briefly introduce the four reasoning formalisms that we study in this paper.

Alloy [11] allows users to create First Order Logic specifications expressed in relational logic. Using the Alloy Analyzer tool, these specifications are grounded within an explicit bounded scope to create a CNF representation. Properties can be expressed as assertions and an off-the-shelf SAT solver, such as Minisat [4], is used to attempt to disprove the assertions by finding counterexamples.

Constraint Satisfaction Problems (CSP) operate over a finite set of variables and a finite set of constraints over them. The CSP solver attempts to find values of all variables so that all constraints are satisfied. For our investigation, we used *Minizinc* [13], a medium-level constraint modeling language and solver designed for specifying constrained optimization and decision problems over integers and real numbers.

Satisfiability Modulo Theory (SMT) solvers combine the standard constraint satisfaction search with richer theories, such as linear arithmetic, bitvectors, arrays, etc. [3]. The standardized input language SMT-LIB2 [2] is often used for modeling the problem. For our investigation, we used Z3, an SMT solver and theorem prover developed at Microsoft Research [3].

Answer Set Programming (ASP) [1] combines a rich declarative input language based on logic programming with negation-as-failure. Solutions to an ASP program are minimal sets of atoms that are supported by and consistent with the program. Modern ASP solvers are based on a modified DPLL procedure or compile the input program into CNF and use an off-the-shelf SAT solver. For our investigation, we used the solver Clasp [9] with the program grounder GrinGo [8]. GrinGo accepts rules with variables, arithmetic, sets, and cardinality constraints.

3. ENCODING MAVO

We introduced FOL semantics for MAVO in [17]. Here, we present an encoding of MAVO models in Relational Algebra (RA), a formalism typically used in the field of database management systems (DBMSs) [18]. This encoding is intended to be used as a specification that can be readily implemented in the same way in *all* four reasoning formalisms that we study in this paper.

Intuitively, our encoding represents MAVO elements as relations whose content is constrained according to their MAVO annotations. We present it by illustrating the steps required to encode the example the partial model M_1 shown in Figure 1(a).

For each type in the MAVO model's type graph, we create an *instantiation relation* which associates MAVO elements of that type with their refining instance elements in the concretization. All instantiation relations for M_1 are shown in Figure 2(b). For example, consider the type `Class` from the type graph G_{CD} in Figure 1(d). Its corresponding instantiation relation, `CLASS`, relates MAVO elements in M_1 with the instance elements in m_1, shown in Figure 1(b), that refine them. For example, `LandVehicle` is refined by the instance elements *Car*, *Truck*, and *Motorcycle*.

Every instantiation relation like `CLASS` has two columns, one for MAVO elements and one for instance elements. We keep track of all elements of the MAVO model in a special relation called `Constants` . The instance elements are declared in a separate set of relations called *universe relations*.

The `Constants` relation has an entry for every MAVO element, keeping track of its name, its type and its MAVO annotations. In our example, the relation `Constants` for M_1 is:

`Constants`:

name	type	**is**M	**is**V	**is**S
`Vehicle`	Class	False	False	False
`LandVehicle`	Class	True	False	True
`X`	Class	False	True	False
`numDoors`	Attribute	False	False	False
`g1`	Inherits	True	False	True
`c1`	Owns	False	False	False

We create one universe relation for each type of element. When a MAVO model is concretized, the universe relations are popu-

(a)

Class_Univ:

Instance element
Vehicle
LandVehicle/Car
LandVehicle/Truck
LandVehicle/Moto

Attribute_Univ:

Instance element
numDoors

Inherits_Univ:

Instance element
g1/1
g1/2

Owns_Univ:

Instance element
c1

(b)

CLASS:

MAVO element	Instance element
Vehicle	*Vehicle*
LandVehicle	*LandVehicle/Car*
LandVehicle	*LandVehicle/Truck*
LandVehicle	*LandVehicle/Moto*
X	*Vehicle*

ATTRIBUTE:

MAVO element	Instance element
numDoors	*numDoors*

INHERITS:

MAVO element	Instance element
g1	*g1/1*
g1	*g1/2*

OWNS:

MAVO element	Instance element
c1	*c1*

(c)

Inherits_Source:

left	right
g1/1	*LandVehicle/Car*
g1/2	*LandVehicle/Truck*

Inherits_Target:

left	right
g1/1	*Vehicle*
g1/2	*Vehicle*

Owns_Source:

left	right
c1	*Vehicle*

Owns_Target:

left	right
c1	*numDoors*

Figure 2: Relational representation of m_1, shown in Figure 1(b): (a) universe relations, (b) instantiation relations, (c) graph relations.

lated with instance elements. The universe relations for M_1 are shown in Figure 2(a). In our example, they have been populated with the instance elements of the concretization m_1. E.g., the relation Inherits_Univ contains two inheritance instance elements, *g1/1* and *g1/2*, as both Car and Truck inherit from Vehicle in m_1.

We define *bound* to be the maximum number of instance elements that can be associated with some MAVO element in an instantiation relation. Choosing an appropriate bound is necessary for most solvers because they typically ground the encoding to some finite representation. The choice of the bound can have significant implications: for example, when the bound is 2, m_1 is not a valid concretization of M_1 because LandVehicle is associated with three instance variables in the relation CLASS.

The relational-algebraic definitions of the instantiation relations of M_1 are shown in lines 3-7 of Figure 3. The instantiation relation for each type is defined as the Cartesian product of its universe with the subset of the elements of that type in the relation Constants . The result is stripped of the additional metadata columns from Constants using projection, to only keep a column for MAVO elements and one for instance elements. In general, in instantiation relations, a MAVO element can be associated with any number of instance elements. Additionally, different MAVO elements may be associated with the same instance element. In our example, in CLASS, LandVehicle is associated with three instance elements (*Car*, *Truck* and *Motorcycle*), while Vehicle and X are both associated with the instance element *Vehicle*.

The possible ways that MAVO elements can be associated with instance elements are constrained by the presence or absence of MAVO annotations. In an instantiation relation, MAVO elements that aren't annotated with S *cannot be associated with more than one* instance element. In our example, only the MAVO elements LandVehicle and g1 are S-annotated, so all others are associated with at most one instance element. The relational-algebraic definitions of the Sconstraints for M_1 are in lines 13-17 of Figure 3. There is one constraint for each MAVO element that does not have an Sannotation. For example, the constraint for the element Vehicle selects the records containing it from its type's instantiation relation (CLASS) and checks whether the selection contains more than one record.

In an instantiation relation, MAVO elements that are not annotated with M *must be associated with at least one* instance element (which is true in our example). Yet g1 is associated with only two inheritance instance elements: there exists one instance of LandVehicle (the element *Motorcycle*) that does not inherit Vehicle. The relational-algebraic definitions of the M constraints for M_1 are given in lines 8-12 of Figure 3. For example, the constraint for the element Vehicle selects the records containing it from its type's instantiation relation (CLASS) and checks whether the selection is empty.

Pairs of MAVO elements of the same type where both don't have the annotation V, e.g., Vehicle and LandVehicle, *cannot share instance elements*. However, Vehicle does share an instance element with the V-annotated element X. The relational-algebraic definitions of the V constraints for M_1 are given in lines 18-23. For example, the constraint for the element Vehicle creates two selections: one by selecting from CLASS all instance elements refining Vehicle and one by selecting from CLASS all instance elements refining other non-V-annotated elements. The constraint then uses natural join to check whether these two selections intersect.

In addition to the universe and instantiation relations, and the MAVO constraints, we also define other relations and constraints, to ensure that concretizations are structurally well-formed graphs. For every *edge* type, we create two *graph relations*, one for the source and one for the target. The graph relations for M_1 are shown in Figure 2(c). E.g., for the type Owns from the type graph G_{CD} in Figure 1(d), we have the relations Owns_Source and Owns_Target. The former says that the source node of *c1* is the class *Vehicle*; the latter that its target node is the attribute *numDoors*.

The relational-algebraic definitions of the graph relations of M_1 are shown in lines 24-28 of Figure 3. According to G_{CD}, the source node of any instance element of type Owns must be of type Class and its target of type Attribute. Therefore, the source graph relation for Owns is defined as the Cartesian product of the universe relation of Owns and that of Class. The target graph relation is defined similarly. For clarity, the columns of each Cartesian product are renamed to "left" and "right", where "left" always refers to the Owns element and "right" to its source or target node.

The well-formedness of concretizations is ensured by constraints over graph relations. We create one graph constraint for each MAVO edge element requiring that (a) if an instance edge exists, its source

1 **Universe relations:**
2 `Class_Univ`, `Attribute_Univ`, `Inherits_Univ`, `Owns_Univ`
3 **Instantiation Relations:**
4 $\texttt{CLASS} = \pi_{\text{MavoElement, InstanceElement}}((\sigma_{\text{type=}\texttt{Class}}\texttt{Constants})\times\texttt{Class\_Univ})$
5 $\texttt{ATTRIBUTE} = \pi_{\text{MavoElement, InstanceElement}}((\sigma_{\text{type=}\texttt{Attribute}}\texttt{Constants})\times\texttt{Attribute\_Univ})$
6 $\texttt{INHERITS} = \pi_{\text{MavoElement, InstanceElement}}((\sigma_{\text{type=}\texttt{Inherits}}\texttt{Constants})\times\texttt{Inherits\_Univ})$
7 $\texttt{OWNS} = \pi_{\text{MavoElement, InstanceElement}}((\sigma_{\text{type=}\texttt{Owns}}\texttt{Constants})\times\texttt{Owns\_Univ})$
8 **M constraints:**
9 $\sigma_{\text{MavoElement=Vehicle}}\texttt{CLASS} \neq \emptyset$
10 $\sigma_{\text{MavoElement=X}}\texttt{CLASS} \neq \emptyset$
11 $\sigma_{\text{MavoElement=numDoors}}\texttt{ATTRIBUTE} \neq \emptyset$
12 $\sigma_{\text{MavoElement=c1}}\texttt{OWNS} \neq \emptyset$
13 **S constraints:**
14 $|\sigma_{\text{MavoElement=Vehicle}}\texttt{CLASS}\ | \leq 1$
15 $|\sigma_{\text{MavoElement=X}}\texttt{CLASS}\ | \leq 1$
16 $|\sigma_{\text{MavoElement=numDoors}}\texttt{ATTRIBUTE}\ | \leq 1$
17 $|\sigma_{\text{MavoElement=c1}}\texttt{OWNS}\ | \leq 1$
18 **V constraints:**
19 $((\sigma_{\text{MavoElement}\neq\text{Vehicle} \wedge \text{isV=False}}\texttt{Constants}) \bowtie_{\text{MavoElement}} \texttt{CLASS}) \bowtie_{\text{InstanceElement}} \sigma_{\text{MavoElement=Vehicle}}\texttt{CLASS} = \emptyset$
20 $((\sigma_{\text{MavoElement}\neq\text{LandVehicle} \wedge \text{isV=False}}\texttt{Constants}) \bowtie_{\text{MavoElement}} \texttt{CLASS}) \bowtie_{\text{InstanceElement}} \sigma_{\text{MavoElement=LandVehicle}}\texttt{CLASS} = \emptyset$
21 $((\sigma_{\text{MavoElement}\neq\text{numDoors} \wedge \text{isV=False}}\texttt{Constants}) \bowtie_{\text{MavoElement}} \texttt{ATTRIBUTE}) \bowtie_{\text{InstanceElement}} \sigma_{\text{MavoElement=numDoors}}\texttt{ATTRIBUTE} = \emptyset$
22 $((\sigma_{\text{MavoElement}\neq\text{g1} \wedge \text{isV=False}}\texttt{Constants}) \bowtie_{\text{MavoElement}} \texttt{INHERITS}) \bowtie_{\text{InstanceElement}} \sigma_{\text{MavoElement=g1}}\texttt{INHERITS} = \emptyset$
23 $((\sigma_{\text{MavoElement}\neq\text{c1} \wedge \text{isV=False}}\texttt{Constants}) \bowtie_{\text{MavoElement}} \texttt{OWNS}) \bowtie_{\text{InstanceElement}} \sigma_{\text{MavoElement=c1}}\texttt{OWNS} = \emptyset$
24 **Graph Relations:**
25 $\texttt{Inherits\_Source}=\rho_{\text{left/}\texttt{Inherits\_Univ}\text{.InstanceElement, right/}\texttt{Class\_Univ}\text{.InstanceElement}}(\texttt{Inherits\_Univ}\times\texttt{Class\_Univ})$
26 $\texttt{Inherits\_Target}=\rho_{\text{left/}\texttt{Inherits\_Univ}\text{ .InstanceElement, right/}\texttt{Class\_Univ}\text{.InstanceElement}}(\texttt{Inherits\_Univ}\times\texttt{Class\_Univ})$
27 $\texttt{Owns\_Source}=\rho_{\text{left/}\texttt{Owns\_Univ}\text{.InstanceElement, right/}\texttt{Class\_Univ}\text{.InstanceElement}}(\texttt{Owns\_Univ}\times\texttt{Class\_Univ})$
28 $\texttt{Owns\_Target}=\rho_{\text{left/}\texttt{Owns\_Univ}\text{ .InstanceElement, right/}\texttt{Attribute\_Univ}\text{.InstanceElement}}(\texttt{Owns\_Univ}\times\texttt{Attribute\_Univ})$
29 **Graph constraints:**
30 $\sigma_{\text{MavoElement=g1}}\texttt{INHERITS} \neq \emptyset \Rightarrow (\sigma_{\text{MavoElement=LandVehicle}}\texttt{CLASS} \neq \emptyset) \wedge (\sigma_{\text{MavoElement=Vehicle}}\texttt{CLASS} \neq \emptyset) \wedge$
31 $\quad(\sigma_{\text{left=g1} \wedge \text{right=LandVehicle}}\texttt{Inherits\_Source} \neq \emptyset) \wedge (\sigma_{\text{left=g1} \wedge \text{right=Vehicle}}\texttt{Inherits\_Target} \neq \emptyset)$
32 $\sigma_{\text{MavoElement=c1}}\texttt{OWNS} \neq \emptyset \Rightarrow (\sigma_{\text{MavoElement=X}}\texttt{CLASS} \neq \emptyset) \wedge (\sigma_{\text{MavoElement=numDoors}}\texttt{ATTRIBUTE} \neq \emptyset) \wedge$
33 $\quad(\sigma_{\text{left=c1} \wedge \text{right=X}}\texttt{Owns\_Source} \neq \emptyset) \wedge (\sigma_{\text{left=c1} \wedge \text{right=numDoors}}\texttt{Owns\_Target} \neq \emptyset)$

Figure 3: Relational encoding of the MAVO model M_1 shown in Figure 1(a).

and target instance nodes also exist, and (b) the edge's type is respected. For example, the existence of the `Owns` instance element c_1 implies the existence of m_1 nodes *Vehicle* and *numDoors*. The type of c_1 is respected because the source and target nodes are included in its corresponding graph relations. The graph constraints for M_1 are shown in lines 29-33 of Figure 3. For example, the constraint for `Owns` mandates that its M-constraint implies the M-constraints of its endpoints (`X` and `numDoors`) and also that its graph constraints contain instance elements of the endpoints.

Implementation details. The RA encoding that we present here is designed to function as an intermediate representation between the FOL semantics of MAVO, presented in [17], and reasoning formalisms such as Alloy, CSP, SMT, and ASP. More efficient encodings, that take advantage of the intricacies of each of these formalisms, can definitely be created. Yet the advantage of the one presented here is that it can be readily translated into each of the four formalisms.

In the CSP encoding, we represent instance elements as integers and the various relations as finite sets of integers. The maximum size of the sets has to be defined statically, depending on the explicit value of bound. The MAVO M- and S-constraints become constraints over the cardinalities of sets, whereas the V-constraints are constraints over the intersection of sets.

In the SMT encoding, we implement instance elements as *abstract values* and relations as *uninterpreted boolean functions*. A function implementing a relation returns True for tuples that belong to the relation, and False otherwise. MAVO constraints are implemented in quantified logic over the truth tables of the functions. Unlike the other formalisms, SMT does not require an explicitbound because the solver is able to efficiently handle infinite types using abstraction.

In the Alloy encoding, we implement relations as Alloy *signatures*. The tool natively supports relational logic, and running it populates the signatures with *atoms* which represent the instance elements. MAVO constraints are expressed as quantified predicates over the signatures. As with the CSP encoding, a bound on the number of instance variables is required.

In the ASP encoding, we use a set of program rules to generate ASP atoms representing a bounded number of instance elements. Another set of program rules generates ASP atoms to represent the relations on these instance variables. Finally, a set of program rules eliminates solutions that do not correspond to valid a concretization given the MAVO constraints.

4. EXPERIMENTS

To study the effectiveness of the four reasoning formalisms for checking properties of MAVO models, we conducted a series of experiments. To simplify our investigation, we did not consider the *OW* partiality type, focusing instead on the other three: *May*, *Set* and *Var*.

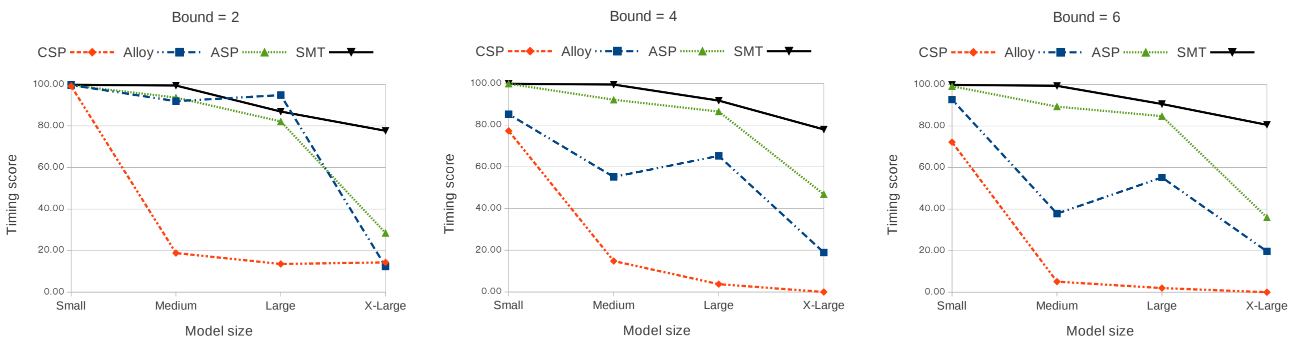

Figure 5: Experimental results.

Size of Model	S	M	L	XL
#Elements	(0, 25]	(25, 50]	(50, 75]	(75, 100]
Exemplar	12	37	62	87
#Nodes	7	14	20	24
#Edges	5	23	42	63

Table 1: Model size categories.

- There exists a node with a self-loop.
- All nodes have outgoing edges.
- All nodes have outgoing or incoming edges.
- For all pairs of nodes $\langle n_1, n_2 \rangle$ there exists at most one edge e such that $n_1 \xrightarrow{e} n_2$
- For every pair of nodes $\langle n_1, n_2 \rangle, n_1 \neq n_2$ there exist two edges $\langle e_1, e_2 \rangle$ such that $n_1 \xrightarrow{e_1} n_2$ and $n_2 \xrightarrow{e_2} n_1$.

Figure 4: Properties used in the experiments.

Experimental Setup. To get inputs for our experiments, we created a random model generator. The generated models conform to a bare-bones metamodel for directed graphs and are randomly decorated with MAVO annotations. Given that solvers typically take advantage of constraints to prune the search space, we opted to using a minimal metamodel of plain directed graphs. Since it imposes a minimal set of constraints, it is thus the most difficult and the most general metamodel for checking properties.

We aimed to create parameterized random models to enable increasing model sizes and levels of uncertainty. To avoid a combinatorial explosion of possible setups, we fixed a few structural parameters of the generated models, based on three "real" case studies constructed using MAVO [17, 16, 10]. In particular, we fixed the graph density of the models (0.11), the overall percentage of MAVO-annotated elements (36%) and the percentages (out of all MAVO-annotated elements) of M-, S- and V-annotated elements (48%, 33% and 43%, respectively).

We discretized the space of possible model sizes by defining four size categories (Small, Medium, Large and Extra Large) which range between 0 and 100 elements such that two ([6, 17]) of the three case studies fell into the Medium category and one ([16]) into the L category. We selected the median number of elements as the exemplar model size for each category (see Table 1). With fixed percentages of MAVO annotations, each category represents an increasingly more difficult verification problem, since the corresponding MAVO model is bigger and further encodes a larger set of concretizations.

Each generated model was automatically translated into each reasoning formalism and checked for five properties. The properties, shown in Figure 4, were inspired from the structural well-formedness constraints of the models in our three reference case studies. The result of the run is determined as follows [6]: a property is True (False) for a MAVO model iff it is True (False) for all its concretizations. If the property is True for some concretizations and False for others, then the result of checking the property is *Maybe*, meaning that it depends on how the MAVO model is going to be refined. Therefore, in order to check a property, we have to run the solver twice: once to find a concretization where the property is True and once where it is False.

Each experiment was repeated for three different values of bounds, set to 2, 4 or 6. We ran each experiment five times and recorded the average for each datapoint. We set cut-off values for runtime and memory consumption to 10 minutes and five GB of RAM, respectively. The experiments were run on an Ubuntu 10.04 LTS 64-bit machine with two quad-core Intel E5355 CPUs with 2.66 GHz and 28GB of RAM. Our recorded observations are available online at `http://www.cs.toronto.edu/~pooya/modevva12.html` .

Results. In our experiments, 8.2% of property checks turned out to be True, 18.4% False, and 32.3% Maybe. An additional 41.1% of checks was inconclusive because of time-outs. We present the (averaged) results in Figures 5(a-c), for bounds 2, 4 and 6, respectively. Each chart tracks the changes in runtime for each modeling formalism as the reasoning problems become harder with increasing size. The horizontal axis captures the model size (S, M, L, XL); the vertical axis records the timing score for each solver. The score is calculated as the percentage of the total allocated time (120 sec.) that was unused after the solver completed all five property checks. A solver that times out for all 5 properties gets a score of 0%. The runtime also reflects the solver's performance with respect to memory consumption (exceeding the memory limit causes disk usage thus slowing the solver down). In general, a higher score means that the solver needed less time and less memory to complete.

In the figures, SMT is represented by a solid black line and is consistently above roughly 80% and (as expected) is unaffected by bound. CSP, indicated by an orange dotted line with rhombuses, consistently scores below 20% for anything except S models. Alloy (blue dotted line with rectangles) performs well for S, M and L models and small bounds, but rapidly deteriorates with larger models and increasing the bound. ASP is indicated by a green dotted line with triangles. Its performance follows that of SMT, but deteriorates for XL models.

Our observations indicate that with increasing bound and model size, SMT generally performs better than the other formalisms. The only case where SMT does not score the best is for models in the L category and bound 2, where it scores 86.92%, compared to 94.83% from Alloy. For the easiest problem (S models and bound 2), all formalisms performed very well, scoring close to 100%. In general, the performance of ASP was comparable to that of SMT, except for models in the XL category. For all bounds and for all categories except XL, the average difference between the score of ASP and that of SMT was 4.42%. However, for XL models their average difference was 41.61%.

The worst score that we observed for SMT was 77.65% for models in the XL category. To further study the limitations of using SMT for reasoning, we conducted an additional experiment. We incrementally increased the size of the problem, extrapolating linearly from the four model size categories in Table 1. We found that SMT's score dropped to 41% for models in the category with (175,200] elements. In other words, doubling the size of the model almost halved the score.

We thus concluded that SMT is the most efficient of the four solvers for checking properties of MAVO models. SMT is unaffected by increases in bound; more importantly, it is designed for reasoning with specifications at a higher level of abstraction, without needing the potentially expensive translation and/or grounding phase required by the other formalisms. These characteristics allow the SMT solver to consistently outperform other formalisms.

Threats to Validity. There are three main threats to the validity of this experimental study: (a) the use of randomly generated MAVO models, (b) the fairness of comparison between the different formalism vis-a-vis the efficiency of the encodings, and (c) our choice of specific reasoning engines. To mitigate the first threat, we tuned the generator to create random models with realistic graph density and frequency of MAVO annotations. Additionally, we fixed these properties to values found in existing MAVO case studies.

With regard to the second threat, it is clear that the perfect comparison would require using the most efficient encoding in each formalism which is impossible to do. Instead, we opted to create a *common* encoding that is directly implementable in each formalism. This way we leveled the playing field, enabling meaningful comparisons.

To adress the third threat, we used solvers that won recent competitions in their respective communities (Z3, Clasp, Alloy with Minisat). To our best knowledge, there has been no recent CSP competition. We thus chose Minizinc because of its convenient modeling language.

5. RELATED WORK

In [6], we experimentally studied the effectiveness of reasoning (property checking and diagnosis) for partial models that only contain May partiality (these are called May models). For this, we used an intuitive encoding of May models directly as propositional expressions which was used as input to a SAT solver. In [15, 7], we used Alloy [11] to verify properties of transformations of MAVO models. In this paper, we expand the scope of our study by considering more types of partiality, as well as by considering different verification tools.

Software product line (SPL) analysis has connections to our work: they express sets of models in a manner similar to May models. In [14], Pohl et. al. do a study to compare the use of SAT, CSP and Binary Decision Diagrams (BDDs) to check properties of feature models. They conclude that BDD-based solvers – the approach we did not examine – have the best performance. Their results are not comparable to ours since they check properties of the entire set of concretizations such as "does their exist a concretization?" whereas we check properties that could hold for each concretization separately (see Figure 4). However, we are inspired to conduct experiments with BDD-based solvers in the future.

6. CONCLUSION

We presented a comparison of four reasoning formalisms (Alloy, CSP, SMT and ASP) with regard to their effectiveness in checking properties of MAVO models. In order to have meaningful comparisons, we introduced an encoding of MAVO, in relational algebra, that can be readily translated into each of these formalisms. We carried out the comparison by running experiments where we used the four reasoning formalisms to check five properties on randomly generated models. Our investigation indicates that, in general, the most efficient formalism is SMT, as it scales better for increasingly hard problems. In the future, we intend to expand the scope of our study to include the *OW* partiality, as well as to incorporate more complex properties, such as those requiring transitive closure.

7. REFERENCES

[1] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, UK, 2003.

[2] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In *Proc. of SMT'10*, 2010.

[3] L. De Moura and N. Bjørner. Satisfiability Modulo Theories: Introduction and Applications. *Commun. ACM*, 54(9):69–77, Sept. 2011.

[4] N. Eén and N. Sörensson. An Extensible SAT-Solver. In *Theory and Applications of Satisfiability Testing*, pages 333–336. Springer, 2004.

[5] M. Famelis, S. Ben-David, M. Chechik, and R. Salay. Partial Models: A Position Paper. In *Proc. of MoDeVVa'11*, pages 1–6, 2011.

[6] M. Famelis, M. Chechik, and R. Salay. Partial Models: Towards Modeling and Reasoning with Uncertainty. In *Proc. of ICSE'12*, June 2012.

[7] M. Famelis, M. Chechik, and R. Salay. The Semantics of Partial Model Transformations. In *Proc. of MiSE'12*, June 2012.

[8] M. Gebser, R. Kaminski, A. König, and T. Schaub. Advances in *gringo* Series 3. In *Proc. of LPNMR'11*, volume 6645 of *LNCS*, pages 345–351, 2011.

[9] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-Driven Answer Set Solving. In *Proc. of IJCAI'07*, pages 386–392, 2007.

[10] J. Gorzny, R. Salay, and M. Chechik. Change Propagation Due to Uncertainty Change, August 2012. submitted.

[11] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[12] V. W. Marek and M. Truszczynski. Stable Models and an Alternative Logic Programming Paradigm. *CoRR*, cs.LO/9809032, 1998.

[13] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack. Minizinc: Towards a Standard CP Modelling Language. In *Proc. of CP'07*, pages 529–543. Springer, 2007.

[14] R. Pohl, K. Lauenroth, and K. Pohl. A Performance Comparison of Contemporary Algorithmic Approaches for Automated Analysis Operations on Feature Models. In *Proc. of ASE'11*, pages 313–322. IEEE Computer Society, 2011.

[15] R. Salay, M. Chechik, and J. Gorzny. Towards a Methodology for Verifying Partial Model Refinements. In *Proc. of VOLT'12*, April 2012.

[16] R. Salay, M. Chechik, and J. Horkoff. Managing Requirements Uncertainty with Partial Models. In *Proc. of RE'12*, 2012. to appear.

[17] R. Salay, M. Famelis, and M. Chechik. Language Independent Refinement using Partial Modeling. In *Proc. of FASE'12*, 2012.

[18] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 2010.

[19] E. Tsang. *Foundations of constraint satisfaction*. Academic Press, London San Diego, 1993.