Agile Validation of Higher Order Transformations using F-Alloy

Loïc Gammaitoni*, Pierre Kelsen† and Qin Ma‡
University of Luxembourg
Email: *loic.gammaitoni@uni.lu, †pierre.kelsen@uni.lu, ‡qin.ma@uni.lu

***Abstract*—Model transformations play a key role in model driven software engineering approaches. Validation of model transformations is crucial for the quality assurance of software systems to be constructed. The relational logic based specification language Alloy and its accompanying tool the Alloy Analyzer have been used in the past to validate properties of model transformations. However Alloy based analysis of transformations suffers from time complexity and scalability issues. The problem becomes even more severe when it comes to higher order transformations that are inherently more complex. In previous work, we proposed a sub-language of Alloy, called F-Alloy, that is tailored for model transformation specifications. Instead of pure analysis based validation, F-Alloy speeds up the validation of model transformations by applying a hybrid strategy that combines analysis with interpretation. In this paper, we show how the F-Alloy approach can be extended to also support efficient validation of higher order transformations.**

I. Introduction

Alloy [1] is a formal specification language based on first-order relational logic. It was developed to support agile modeling of software designs. Alloy models can be automatically analyzed by the Alloy Analyzer to check the correctness of software designs. By providing immediate feedback to users, the use of Alloy is meant to facilitate identifying design errors early in software development life cycle.

In the context of model driven software engineering, Alloy and its accompanying tool the Alloy Analyzer have been used to validate properties of models [2], [3] and model transformations [4], [5]. However, the analysis of specifications expressed in Alloy has some important limitations. Alloy employs a SAT based backend and performs bounded exhaustive search during analysis. The effective execution of an Alloy analysis heavily relies on the specification of the bounds, called scopes in Alloy, which limit the number of elements of each type. The Alloy Analyzer populates model instances with numbers of elements of a given type up to the given scope and performs an exhaustive search in this finite space of model instances to find a solution or a counter-example. As a consequence, two major problems hamper the practical application of Alloy:

1) Despite many advances in the performance of SAT solvers, the analysis can still become quite time consuming especially when the model requires a larger scope to find a suitable instance, increasing scopes leading to a combinatorial explosion
2) The task of finding a minimal sufficient scope is by itself non-trivial (in fact it is undecidable). This is particularly problematic for complex Alloy specifications such as those for expressing higher order model transformations

In previous work [6], we proposed a sub-language of Alloy, called F-Alloy, that is tailored for model transformation specifications. Instead of pure analysis based validation, F-Alloy has the potential to speed up the validation of model transformations by applying a hybrid strategy that combines analysis with interpretation. In this paper, we coin the notion of "hybrid analysis" of model transformations in Alloy and explain how the interpretation of F-modules (modules written in F-Alloy) can be seamlessly integrated with the SAT-based analysis of Alloy modules to realize this approach. Moreover, we show how the F-Alloy approach can be extended to also support efficient validation of higher order transformations.

The rest of the paper is structured as follows. In Sect. II, we introduce the Alloy specification language and its accompanying analysis tool. In Sect. III, we review F-Alloy for transformation specifications and extend the language to also support higher order transformations. We define the hybrid analysis strategy in Sect. IV by integrating Alloy's analysis with F-Alloy's interpretation, and we evaluate our approach in Sect. V. We discuss related work in Sect. VI and present concluding remarks and future work in the final section.

II. Alloy and The Alloy Analyzer

Alloy [1] is a textual modelling language based on first order logic and relational calculus. It combines the precision of formal specifications with powerful automatic analysis features offered by the accompanying analyzer tool.

A. Metamodels in Alloy

A metamodel consists in Alloy of an *Alloy module* (that may import other Alloy modules), an Alloy module being a file containing Alloy specifications. Alloy modules are composed of *signatures* (similar to the notion of "classes" in metamodels) and *facts* (similar to the notion of invariants in metamodels). A signature **sig** A{} defines a basic type A and represents a set of A-*atoms*. The **extends** keyword can be used in signature declarations to introduce subtypes (and consequently subsets of atoms) of another signature. Fields can be declared inside signatures to define relations over sets of atoms. For example **sig** A{f: B −> C} defines a ternary relation $f \subseteq A \times B \times C$. In addition, *functions* and *predicates* can be declared to define named parameterized relational or boolean expressions that

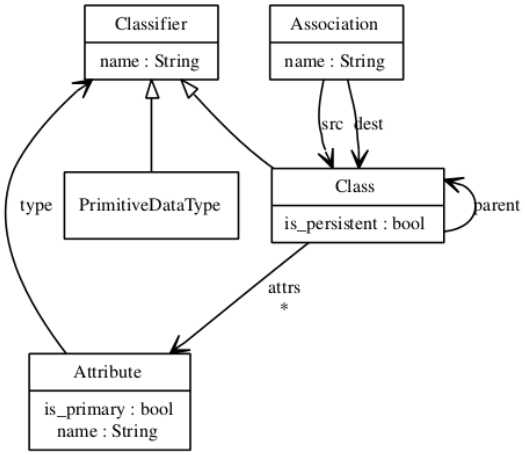

Fig. 1. Class Diagram (CD) metamodel (adapted from [7])

can be invoked in other places. Finally *assertions* are named constraints that can be checked to look for counterexamples.

As an example, the class diagram (CD) metamodel depicted in fig. 1 could be represented in an Alloy module by the following snippet (including well formedness constraints not depicted in fig. 1).

```
abstract sig CDElem{
// each CDElem has different name
  disj name: String
}
abstract sig Classifier extends CDElem{}
sig PrimitiveDataType extends Classifier{}
 sig Class extends Classifier{
  attrs: set Attribute, // * multiplicity
  parent: lone Class, // 0,1 multiplicity
  is_persistent: Bool
}{
  is_persistent = True implies parent = none
  is_persistent = True implies some a: attrs|a.
       is_primary = True
}
fact noCycle{
// no class c in the transitive closure of c.parent
  no c: Class|c in c.^parent
}
sig Attribute extends CDElem{
  is_primary: Bool,
  type: Classifier
}{
  this in Class.attrs
  is_primary = True implies type in PrimitiveDataType
  type in Class implies type.is_persistent = True
}
sig Association extends CDElem{
  src: Class,
  dest: Class
}{
  src ≠ dest
 }
 run {} for 5
```

Listing 1. Class diagram metamodel specified in Alloy

A model *conforming* to an Alloy module a is a set of atoms (typed by signatures of a) and tuples of atoms (typed by fields of a) so that all the constraints (mainly expressed through facts) of a are satisfied. In the Alloy community, those conforming models are called *instances*. We further call a-*instances* models conforming to a.

The run command, *i.e.*, `run {} for 5` at the end of the example above, instructs the Alloy Analyzer to search for all instances that satisfy the specification within the given scope. In the above example, the scope is set to at most 5 atoms for all top-level signatures (i.e., signatures that do not extend another one). The effectiveness of the Alloy based validation relies on the small scope hypothesis which basically claims that most design errors can be found in small counterexamples.

B. Model Transformations in Alloy

Alloy modules can also be exploited to specify model transformations [4], [5]. In this paper we focus on model transformations that take one source model as input and produce one target model as output. The source model conforms to a source metamodel and the target model conforms to a target metamodel. Suppose the source metamodel is specified in an Alloy module a_1 and the target metamodel in an Alloy module a_2. The Alloy module a representing the transformation *imports* both a_1 and a_2, and specifies a set of relations and predicates to express the transformation rules.

The Alloy Analyzer analyzes the Alloy module a and searches (within a scope) for all pairs of instances of a_1 and a_2 such that the relations and predicates corresponding to the transformation rules hold. Each pair of instances represents an input model of the transformation and the corresponding output model. The idea of using Alloy for specifying model transformations is appealing. In addition to automatic input and output model instantiation, one can also exploit Alloy for various validation tasks such as the validity of output models, properties of input-output model pairs, and so on. However, depending on the size of a_1 and a_2, and the scope, the analysis can become very resource consuming and in theory even undecidable. For example, the Alloy Analyzer fails to analyze the class diagram (CD) to relational database management systems (RDBMS) transformation example [7] when the scope is 20, on a computer with a Quad-Core Intel i7 CPU and 8 GB memory. A memory overflow error is reported before a result can be found.

In order to fully leverage the validation power offered by Alloy within reasonable computing resources, in a previous work [6], we proposed a sub-language of Alloy, called F-Alloy, that is tailored for model transformation specifications and efficient model transformation interpretation.

III. An Interpretable Model Transformation Language: F-Alloy

A. Basic F-modules

The F-Alloy language [6] is an Alloy-based Domain Specific Language (DSL) for model transformations. F-Alloy is a sub-language of Alloy in the sense that every F-module (i.e., modules expressed in F-Alloy) is syntactically also an Alloy module; semantically, every F-module can be translated into an equivalent plain Alloy module by adding constraints.

To express a model transformation, an F-module f imports the two Alloy modules a_1 and a_2 that represent the input and output metamodel, respectively, and specifies in its body the following.

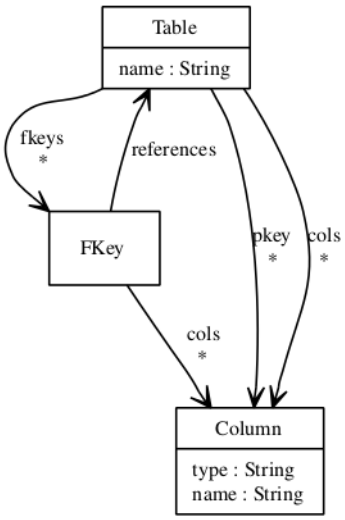

Fig. 2. Relational Database Management System (RDBMS) metamodel (adapted from [7])

- A single *Bridge* signature aggregates all the transformation rules in terms of *mappings*. A mapping relates one or several signatures of a_1 to a signature of a_2 to indicate that atoms typed by the(se) a_1 signature(s) in the input model will be transformed into atoms typed by the a_2 signature in the output model.
- For each mapping (or transformation rule) named r, a corresponding *Guard* predicate named $guard\_r$ can be specified. It defines a constraint that must hold on the input model in order for the transformation rule to take effect.
- Finally, for each mapping (or transformation rule) named r, a corresponding *Value* predicate named $value\_r$ can be specified. It defines a constraint that must hold on the input and output models together after applying the transformation rule.

Compared to expressing model transformations in plain Alloy specifications, F-Alloy offers three advantages.

1) It allows to represent transformations more concisely by relieving the designer of having to write a number of implicit constraints. These constraints are inherent to the semantics of F-modules.
2) Modules expressed in F-Alloy are efficiently interpretable.
3) And finally, with F-Alloy being interpretable, the specification of an appropriate scope becomes irrelevant.

As an example, we look at the CD2RDBMS transformation example where a class diagram (instance of the metamodel defined in Fig. 1) is transformed into a relational database (instance of the metamodel defined in Fig. 2). Briefly, CD2RDBMS transforms persistent top-level classes into tables, flattens the class inheritance hierarchy and transforms all attributes (resp. associations), of the class and those of its sub-classes, into columns. Please refer to [7] for a complete description of the model transformation and to [8] for a full-fledged implementation in F-Alloy.

The following is an excerpt of the CD2RDBMS transformation written in F-Alloy. The full specification can be found in [8].

```
module UML2RDBMS

open CD/AbstractSyntax/CD
open RDBMS/AbstractSyntax/RDBMS

one sig Bridge{
  class2table: Class->Table,
  primAttr2column: Attribute->Column,
  classAttr2column: Attribute->Attribute->Column,
  classAttr2Fkey: Attribute->FKey,
  association2column: Association->Attribute->Column,
  association2FKey: Association->FKey,
}
pred guard_class2table(c: Class){
  c.is_persistent = True c.parent = none
}
pred value_class2table(c: Class, t: Table){
  t.name[0] = c.name
}
......
```

Listing 2. Excerpt of CD2RDBMS transformation specified in F-Alloy

This excerpt contains the Bridge signature in which six mappings are specified. Only the guard and value predicates of the `class2table` mapping are included for illustration purpose, while guard and value predicates of the other mappings are omitted. The guard predicate `guard_class2table` states that only persistent topmost classes are to be mapped to a new table and the value predicate `value_class2table` states that each table should be named after the class which it is mapped to.

B. Higher Order F-Alloy

In [6], it is assumed that an F-module always opens two Alloy modules (Fig. 3 left) thus only basic model transformations are considered. However, higher order transformations play also a key role in various model driven software engineering activities. Indeed, they already have a wide spectrum of applications as reported by a survey on the use of higher order transformations [9]. In this paper we extend the definition of F-modules so that the two opened modules can be either plain Alloy modules or F-modules. Fig. 3 represents this extension in terms of metamodel update.

The recursive definition of F-modules (Fig. 3 right) takes a uniform format for both basic transformations (when the input and output modules are both Alloy modules) and higher order transformations (when either the input or the output module is an F-module). In the following, we use a to range over plain Alloy modules, f to range over F-modules and m to range over both of them. Moreover, we abstract F-modules into the form of $f : m_1 \rightarrow m_2$, where m_1 is the input module and m_2 the output module.

In addition to the syntactical extension, the semantics of F-modules needs to be updated as well. In [6], a translational semantics is given that maps F-modules to semantically equivalent plain Alloy modules referred to as *augmented modules*. We denote the translation procedure by $\mathcal{A}(f : a_1 \rightarrow a_2)$. Note that $\mathcal{A}$ takes only basic F-modules as input, namely the two imported modules a_1 and a_2 are both plain Alloy modules.

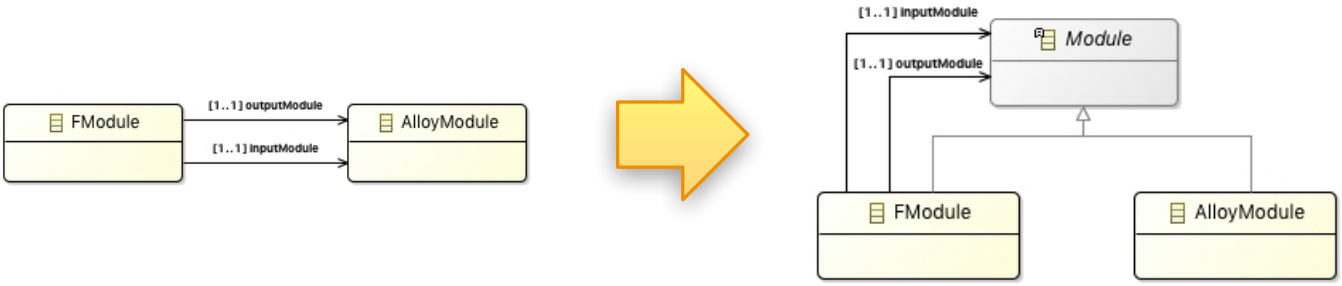

Fig. 3. From Basic F-modules (left) to Higher Order F-modules (right)

Let $[m]_{\mathcal{A}}$ stand for the corresponding augmented module (namely the semantics) of a module m as defined in Fig. 3, right. We have:

$$[f : m_1 \to m_2]_{\mathcal{A}} \stackrel{\text{def}}{=} \mathcal{A}(f : [m_1]_{\mathcal{A}} \to [m_2]_{\mathcal{A}})$$
$$[a]_{\mathcal{A}} \stackrel{\text{def}}{=} a$$

Briefly, before applying the augmentation procedure as defined in [6], the imported modules of an F-module need to be augmented first.

An F-module is interpreted instead of analyzed. F-Alloy interpretation takes two inputs: (1) an F-module f that represents a transformation from m_1 to m_2, and (2) an instance x_1 of m_1. In case m_1 itself is also an F-module, m_1 will be first recursively interpreted to build x_1, then x_1 is provided for the interpretation of f.

Given the inputs, the F-Alloy interpretation will then build, following the mappings in f, an instance x (if such an instance exists) conforming to f. The projection of x onto m_1 is the input instance x_1 and the projection of x onto m_2 is the corresponding output instance. Instance x is built in a backtrack-free fashion as documented in the pseudo-code given below.

```
/*INPUT : m is an F-module from m1 to m2 and x1 is an
     instance of m1
 *OUTPUT: return an instance of m or none */
FUNCTION Interpretation(F-module m, Instance x1)
   b= new Atom(Bridge)
   // x is the m-instance we will return
   x = b // it initially contains a bridge atom
   x += x1 // and atoms and tuples of x1
   FOR EACH bridge mapping r of the form T->S
  //T being a tuple of m1-signatures, S being an m2-
       signature
      FOR EACH T-tuple t in x1
      such that guard_r(t) holds
         a = new Atom(S)
         x += a
         x += new Tuple(b,t,a) of type r
      DONE
   DONE
   FOR EACH bridge mapping r of the form T->S
      FOR EACH T-tuple t in x1
      such that guard_r(t) holds
         x += new Tuple(value_r(t))
      DONE
   DONE
   IF x conforms to m
      RETURN x
   ELSE
      RETURN NONE
END FUNCTION
```

Listing 3. F-Alloy Interpretation pseudo-code

As suggested by the given pseudo-code, F-Alloy interpretation can be performed in polynomial time in terms of the size of the input instance given. The complexity analysis of the F-Alloy interpretation provided in [6] thus still holds.

It is also shown in [6] that an F-module f and its corresponding augmented module $[f]_{\mathcal{A}}$ are *semantically equivalent* in the following sense:

Theorem 1 (Correctness of F-Alloy Interpretation)**.** *Given an F-module f from m_1 to m_2, and its corresponding augmented module $[f]_{\mathcal{A}}$, for any m_1-instance x_1 , the instance produced by the interpretation of f given x_1, is isomorphic under renaming to all the instances obtained by the analysis of $[f]_{\mathcal{A}}$ where the projection of these instances onto m_1 is x_1.*

IV. Hybrid Analysis of (Higher Order) Model Transformations

Instead of pure analysis based validation, F-Alloy has the potential to speed up the validation of model transformations, basic ones and higher order ones, by applying a hybrid strategy that combines the analysis of Alloy with the interpretation of F-Alloy. More specifically, given a module m, the hybrid analysis of m returns a set of instances of m by working as follows.

Definition 1 (Hybrid Analysis)**.** *The set of instances returned by Hybrid Analysis of a module m is:*

- *If m is an F-module from m_1 to m_2: the set of m-instances obtained by:*
  1) *applying the hybrid analysis on m_1*
  2) *applying the F-Alloy interpretation on m for each instance of m_1 obtained in step 1*
- *If m is a plain Alloy module: the set of m-instances obtained by applying Alloy analysis to m .*

Note that the import hierarchy of an F-module, just like any Alloy import hierarchy, is acyclic, which implies the existence of an Alloy module at the source of any higher order transformation expressed in F-Alloy.

We demonstrate the correctness of the hybrid analysis strategy by the following theorem, which basically states that the

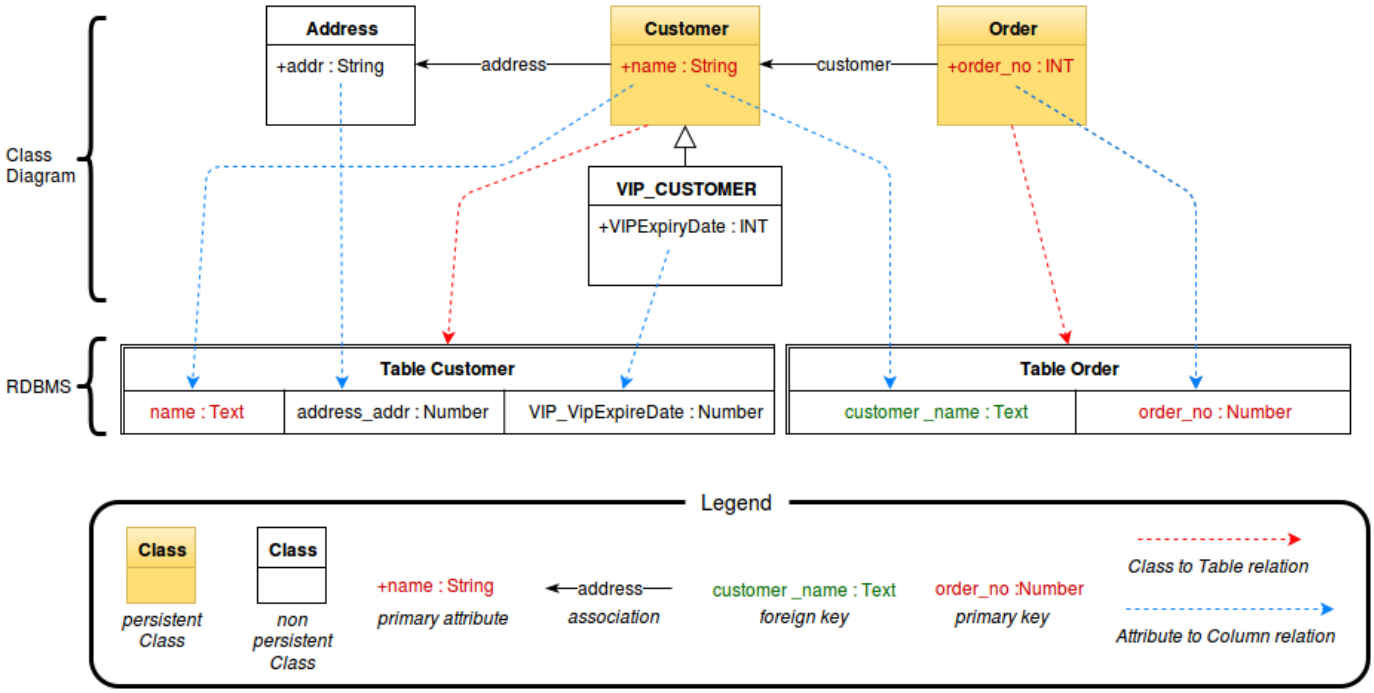

Fig. 4. Visualization of a CD2RDBMS transformation instance produced by CD2RDBMSviz

result obtained by applying the hybrid strategy is equivalent to the result obtained by applying the classic Alloy analysis.

Theorem 2 (Correctness of Hybrid Analysis)**.** *Given an F-module m and its corresponding augmented module $[m]_{\mathcal{A}}$, the set of instances obtained by applying the hybrid analysis on m is isomorphic under renaming to the set of instances obtained by applying Alloy analysis on $[m]_{\mathcal{A}}$.*

Proof. We prove by induction.

- In case m is a plain Alloy module, the theorem holds trivially following the definition of hybrid analysis and theorem 1.
- In case m is an F-module from m_1 to m_2, by induction hypothesis, the set of instances obtained by hybrid analysis on m_1 is equivalent to the set of instances obtained by Alloy analysis on $[m_1]_{\mathcal{A}}$. Following the definition of $[m]_{\mathcal{A}}$ and Theorem 1, the set of instances obtained by interpreting m for each instances of m_1 is also equivalent (up to renaming) to the set of instances obtained by analyzing $[m]_{\mathcal{A}}$ with the Alloy analyzer.

□

We implement the hybrid analysis strategy by the following pseudo code.

```
FUNCTION HybridAnalysis(Module m)
// return set of instances of m
   IF m is a functional module from m1 to m2
      S = HybridAnalysis(m1)
      LET T = emptySet; // set of m-instances
      FOR EACH x in S DO
         y = Interpretation(m,x);
         T = T ∪ {y}
         RETURN T;
      DONE
   ELSE // m is not a functional module
      RETURN AlloyAnalysis(m);
   FI
END FUNCTION
```

Listing 4. Hybrid Analysis pseudo code

Compared to pure Alloy based analysis, the hybrid analysis implementation improves substantially the performance of transformation validation in terms of reduced computation complexity. Let f_n be an F-module with its transformation hierarchy depicted in fig. 5.

Instead of analyzing $[f_n]_{\mathcal{A}}$ using the Alloy Analyzer, which is very resource consuming especially in case of big scopes, one only needs to analyze the left-most leaf Alloy module a_{11} and all the rest can be built by applying the F-Alloy interpretation to $f_1, \ldots, f_n$ within polynomial time. In addition, the problem of finding an optimal scope for a given analysis is also reduced accordingly, where one only needs to define a scope for a_{11} instead of f_n which is much more complex.

More specifically, the time complexity of hybrid analysis is given below.

Proposition 1 (Hybrid Analysis Complexity)**.** *The time needed for hybrid analysis of an F-module f from m_1 to m_2 is a polynomial function of the time needed for the hybrid analysis of m_1.*

In the next section, we apply the hybrid analysis to two case studies, one a basic transformation and the other a higher order transformation, to provide empirical evidence for the previous claim.

V. Evaluation

Alloy analysis is commonly used throughout model driven software development processes for validating models, where

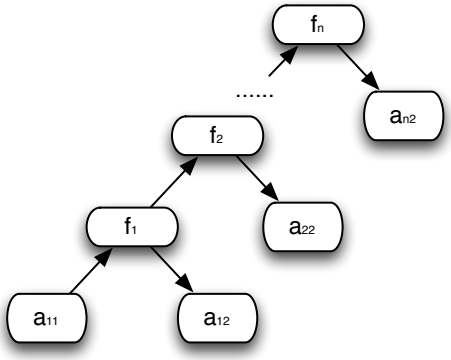

Fig. 5. Import Hierarchy of an F-module

instances of models are automatically generated and reviewed for potential errors. Such a scenario can often be found in case of agile software development where the designer incrementally improves his models after spotting errors in instances obtained by analysis [10].

In this section, we evaluate the efficiency of hybrid analysis by comparing its performance with Alloy analysis. Two case studies are considered: a basic transformation from class diagrams to relational databases (i.e., the CD2RDBMS example introduced in Sect. III-A), and a higher order transformation called CD2RDBMSviz that generates a visualization for CD2RDBMS. For each instance of CD2RDBMS, namely a class diagram and its corresponding relational database counterpart, the visualization produced by CD2RDBMSviz consists of a visual representation of the class diagram instance, a visual representation of its RDBMS counterpart, and a visual representation of the mappings between the two. Fig. 4 shows an example of such a visualization.

Fig. 6 illustrates the hierarchical module structure of the two case studies. According to Definition 1, the hybrid analysis of module CD2RDBMSviz performs as follows:

1) CD2RDBMSviz being an F-module, hybrid analysis is performed on the CD2RDBMS module.
   a) the CD2RDBMS module being an F-module, hybrid analysis is performed on the CD module.
   b) the CD module being an Alloy module, CD-instances are generated using the Alloy analyzer.
   c) For each instance obtained in Step b, an interpretation of the CD2RBDMS module is performed – resulting in a set of CD2RDBMS-instances.
2) For each instance obtained in Step c, an interpretation of the CD2RBDMS2viz module is performed – resulting in a set of CD2RDBMS2viz-instances.

Note that Steps a to c also correspond to the hybrid analysis of module CD2RDBMS.

Time measurements of Alloy and hybrid analysis applied to the CD2RDBMS and CD2RDBMSviz modules, as well as the time needed for the Alloy analysis of the CD module can be found in table I (times given in ms). Note that the symbol ∞ has been used to mark operations that failed to finish.

Measurements show that the time needed to perform an Alloy analysis increases substantially with the scope and the size of the model. They also show that the time needed to find the first instance of CD2RDBMS and CD2RDBMSviz using hybrid analysis, is of the same order of magnitude than the time needed to find the first instance of the CD module,

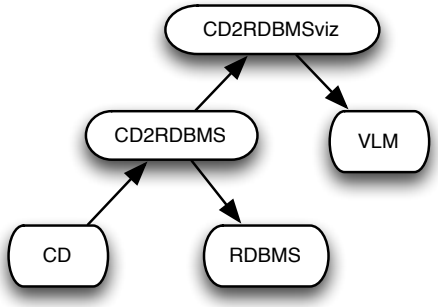

Fig. 6. Module Hierarchy of CD2RDBMS and CD2RDBMSviz

for a given scope. These observations provide us with positive support for Proposition 1.

Note that the hybrid analysis of CD2RDBMS and CD2RDBMSviz surprisingly took less time to complete for a scope of 15 than for a scope of 10. This is due to the fact that the time needed for interpretation to complete is proportional to the size of the instances given to the interpreter and to the fact that the first instance obtained by the analysis of CD with a scope of 10 was bigger than the one obtained with a scope of 15.

VI. Related Work

Our work is related to the general problem of speeding up the analysis of Alloy specifications. Different approaches have been proposed for this in the literature. To understand these approaches, we need to recall how the Alloy Analyzer works. The analysis is based on transforming the Alloy model into a propositional formula that is fed into an off-the-shelf SAT solver.

Alloy analysis can be optimised by improving the transformation from Alloy model to propositional formula. In [11] traditional compiler optimisations are used to improve this transformation, resulting in some cases in time reductions of an order of magnitude.

A different class of optimisation techniques is based on applying slicing techniques to Alloy models. In [12] a sub model is identified, called a base slice, that is simpler and more efficiently solvable. Such a base slice can either be proven unsolvable, or extended in a systematic fashion into a full solution. This second approach is closer to our proposed approach: in our case the extension of a partial instance to a complete instance is done via interpretation, while in [12] it is done using a constrained analysis (guaranteeing that a solution for the whole model satisfies the constraints of the base slice).

Yet another type of approach, described in [13], proceeds by annotating Alloy models with meta-information that allows to use domain-specific solvers (e.g., String and Integer solvers)

Alloy analysis Scope	CD	CD2RDBMS		CD2RDBMSviz	
	Alloy Analysis	Alloy Analysis	Hybrid Analysis	Alloy Analysis	Hybrid Analysis
5	26	1720	95	3078	229
10	53	7251	181	15862	386
15	76	24671	173	∞	337
20	1844	∞	1918	∞	2053

TABLE I
COMPARATIVE TABLE: TIME NEEDED IN MILLISECONDS TO FIND THE FIRST INSTANCE

to solve sub-models and combine the output of those solvers with the SAT-based backend of the Alloy Analyzer.

Rather than improving the speed of the analysis one can attempt to improve its applicability. More specifically, an approach based on SMT solvers allows to drop the finite scope assumption of Alloy's analyzer and actually prove properties of a model regardless of the scope [14]. The drawback of such an approach is the possible need for manual intervention.

VII. CONCLUSION AND FUTURE WORK

This paper presents an application of F-Alloy, which is an Alloy based language for model transformations, to higher order transformations. In addition to leveraging the automatic instance generation facility of Alloy analysis for validation of model transformations, the hybrid analysis strategy proposed in this paper further speeds up the validation process, by combining Alloy analysis with F-Alloy interpretation. We evaluate the efficiency of our approach by applying it to two case studies. According to the experiments, the complexity reduction is substantial, for both basic and higher order transformations.

The combined power of Alloy analysis and F-Alloy interpretation provides us with a lot of potential in terms of formal method based model transformation validation. As far as higher order transformations are considered, in this paper, our approach is only applied to one type of higher order transformations following the classification given in [9], namely "transformation analysis". As future work, we plan to apply our approach to more cases to also cover examples from other types of higher order transformations.

In addition, despite the great increase of efficiency exposed in this paper, one might still shun using F-Alloy due to the complexity of its formal syntax. This is a common critique to formal method based approaches. To make our approach more accessible, we also plan to extend F-Alloy with a user-friendly graphical concrete syntax.

REFERENCES

[1] D. Jackson, *Software abstractions*. MIT Press Cambridge, 2012.

[2] K. Anastasakis, *A model driven approach for the automated analysis of UML class diagrams*. University of Birmingham, 2009.

[3] T. T. Dinh-Trong, S. Ghosh, and R. B. France, "A systematic approach to generate inputs to test UML design models," in *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*. IEEE, 2006, pp. 95–104.

[4] K. Anastasakis, B. Bordbar, and J. M. Küster, "Analysis of model transformations via Alloy," in *Proceedings of the 4th MoDeVVa workshop: Model-Driven Engineering, Verification, and Validation*, 2007, pp. 47–56.

[5] L. Baresi and P. Spoletini, "On the use of alloy to analyze graph transformation systems," in *Proceedings of the 3rd ICGT: International Conference on Graph Transformations*, ser. LNCS 4178, 2006, pp. 306–320.

[6] L. Gammaitoni and P. Kelsen, "F-Alloy: An Alloy based model transformation language," in *Proceedings of the 8th International Conference on Theory and Practice of Model Transformations - ICMT 2015*, ser. LNCS 9152, 2015, pp. 166–180.

[7] J. Bézivin, B. Rumpe, A. Schürr, and L. Tratt, "Model transformations in practice workshop," in *Satellite Events at the MoDELS 2005 Conference*. Springer, 2006, pp. 120–127.

[8] L. Gammaitoni and P. Kelsen, "An F-Alloy specification for the CD2RDBMS case study," http://lightning.gforge.uni.lu/doc/TR-LASSY-15-01.pdf (2015).

[9] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin, "On the use of higher-order model transformations," in *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA 2009*, 2009, pp. 18–33.

[10] L. Gammaitoni, P. Kelsen, and F. Mathey, "Verifying modelling languages using Lightning: a case study," in *Proceedings of the 11th MoDeVVa workshop: Model-Driven Engineering, Verification, and Validation*. Springer, 2014, pp. 19–28.

[11] D. Marinov, S. Khurshid, S. Bugrara, L. Zhang, and M. Rinard, "Optimizations for compiling declarative models into boolean formulas," in *Theory and Applications of Satisfiability Testing*. Springer, 2005, pp. 187–202.

[12] E. Uzuncaova and S. Khurshid, "Constraint prioritization for efficient analysis of declarative models," in *Proceedings of the 15th FM: International Symposium on Formal Methods*. Springer, 2008, pp. 310–325.

[13] S. Ganov, S. Khurshid, and D. E. Perry, "Annotations for Alloy: automated incremental analysis using domain specific solvers," in *Formal Methods and Software Engineering*. Springer, 2012, pp. 414–429.

[14] A. A. El Ghazi and M. Taghdiri, "Analyzing Alloy constraints using an SMT solver: a case study," in *5th International Workshop on Automated Formal Methods (AFM). Edinburgh*, 2010.