2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)

First-Order Quantified Separator in Alloy Analyzer

One An
onean@sas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania, USA

Abstract

First-Order Logic (FOL) is a powerful language for specifying system invariants and properties, yet its formal complexity often hinders its adoption. To address this, we present Folloy, a novel tool that synthesizes FOL specifications from examples. Our core contribution is a new approach that translates the specification learning problem into a constraint satisfaction problem by declaratively modeling FOL's syntax and semantics in the Alloy Analyzer. This method is highly expressive, allowing for the synthesis of non-prenex formulas and user-defined syntactic constraints. By leveraging a Max-SAT solver, Folloy also guarantees that the learned formula is minimal in size. We evaluate our tool on a suite of benchmark problems and show that while this general approach is slower than a specialized algorithm, it solves a broader class of problems, establishing a clear trade-off between performance and expressive power.

CCS Concepts

• **Software and its engineering** → *Formal methods.*

Keywords

invariant inference, first-order logic, Alloy Analyzer

ACM Reference Format:
One An. 2025. First-Order Quantified Separator in Alloy Analyzer. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE '25).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction and Background Work

A specification in *first-order logic (FOL)* details contraints on a system's state; often, invariants, pre- and post-conditions, and relational properties are expressed with FOL. Formal methods tools such as Alloy, TLA+, and Dafny are built upon FOL.

However, writing a specification requires learning formal logic, which has shown to have a steep learning curve [2, 6]. This presents *specification learning from examples* as a potential solution. In this framework, the user would provide positive and negative examples; i.e. structures desired and structures prohibited in the system. The goal is to output a *separator* formula in FOL such that it satisfies the positive examples and contradicts the negative examples.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASE '25, Nov 16-20, 2025, Seoul, South Korea
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

Figure 1 shows an example of a separator where the formula states that every edge must be part of a triangle. Thus, the formula separates the examples marked by + from examples marked by −.

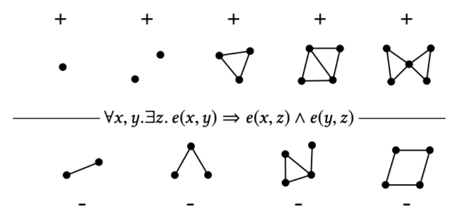

Figure 1: Example of Separator from Koenig et al. [3]

A *k-depth prenex* separator is a formula whose quantifier prefix has length $\leq k$; Koenig et al. show this separability problem is NP-complete and give a fixed-k algorithm [3].

Inspired by Zhang et al [7], this work proposes a method for finding a separator by encoding this problem in Alloy$^{\text{Max}}$, an extension of Alloy Analyzer that uses a Max-SAT solver. This approach also allows for a minimization constraints (such as on the number of symbols), often leading to more legible and simpler formulas. Following their approach, we also focus on the *constrained first-order separability* problem and also allow for additional syntactic constraints on the separator, taking full advantage of Alloy's expressive language. This method is implemented in a new tool, named Folloy.

Folloy has multiple potential applications. It can be used as part of an inductive invariant learning algorithm such as the ICE/IC3 algorithms. Another application is program sketch: Scythe [1] achieves completion of protocol sketches using a CEGIS loop in TLA+. Folloy can potentially fill in holes for Alloy when provided positive and negative instances. Folloy can be also used in specification repair by providing incorrect specification along with examples and using a constraint to keep the new specification structurally as closest possible to the original.

2 Problem Formulation

A model triple $M = (D, R, A)$ where
- $D = \{D_s\}_{s \in S}$ is a set of domains indexed by sorts s in the sort set S. $D ::= \varnothing \mid c_s, D$ is a set of constants.
- $R ::= \varnothing \mid r(c_{s_0}, ..., c_{s_n}), R$ is a set of relations among the constants in the domain.
- $A = \{A_s\}_{s \in S}$ where $A ::= \varnothing \mid x \mapsto c_s, A$ is a set of functions from variables to constants.

The semantics of satisfaction $M \vDash \varphi$ is given by a standard multi-sorted model-theoretic definition such as in [4].

A constrained FOL separability problem is defined by a triple (P, N, Φ) where P and N are set of positive/negative example models and Φ is an FOL formula that constrains the syntactic structure of the separator.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00391

The goal is to find a separator formula φ such that for all $M \in P$, $M \models \varphi$ and for all $M' \in N$, $M' \not\models \varphi$; i.e., a formula that satisfy the positive examples and does not satisfy the negative examples.

3 Alloy Encoding

The syntax is encoded in Alloy as a directed acyclic graph (DAG). We use Alloy **sig**natures to define the language's components, including logical connectives (`And`, `Or`, `Not`), quantifiers (`Forall`, `Exists`), and atomic formulas (`Atom`). A set of structural **facts** constraints all components to form a valid DAG and ensure the variables are bound and scoped to quantifiers correctly.

The semantic encoding implements the formal semantics. A generated formula is constrained such that for all `Environments` and components, the elements within the `model` are a satisfying embedding of the given component.

For example, Figure 2 contains the semantic rule for the `Forall` quantifier. We define that the `Forall` quantifier is satisfied in an `Environment` if and only if for all elements of the model e, the body is true in the `Environment` extended with e for the bound variable of the quantifier. To make semantics of quantifiers correct, the encoding also includes all variable assignments hard-coded.

```
// A "sig" (signature) defines a type of atoms.
abstract sig Model {
    elements: set Element,
    interpretation: Relation -> set Tuple,
    satisfies: Environment -> set Formula }

// "fact" defines constraints for the semantics.
fact Semantics { all s: Model {
    all env: Environment, f: Forall |
        (env -> f) in s.satisfies iff
        (all e: s.elements | e.sort = f.var_sort
            implies
        (one enb: extendEnv[env, f.bound_var, e] |
             (enb -> f.body) in s.satisfies)) }}

one sig Env31 extends Environment {} { mapping =
    V0->E0 + V1->E0 + V2->E0 }
```

Figure 2: Semantic Encoding in Alloy

Examples are translated into a **sig**nature extending the `PosModel` **sig**nature along with the elements of the example. The relations over the example is defined as a fact as shown in Figure 3.

Running the Alloy with the **fact** that the root satisfies the positive examples and does not satisfy the negative examples outputs an instance which is a separator.

4 Results

The evaluation we used is the test suite containing positive and negative examples from Koenig et al. [3]. With reduced number of total examples, Folloy had a success rate of 100% on all 6 problems. However, the average runtime was 6.03 minutes using the SAT4JMax solver compared to Koenig et al.'s 0.3 seconds.

We also compared the generated formula with and without the minimization clause for the number of symbols. With the exception

```
abstract sig PosModel extends Model {}

one sig PS0 extends PosModel{} {
        elements = E0 + E1  }

fact PS0Constraints {
        some t: PS0.interpretation[edgeRel] | t.
            tup[I0] = E0 and t.tup[I1] = E1 }
```

Figure 3: Examples Encoded in Alloy

of one problem, adding the minimization found a smaller formula. Given that a FOL separability problem may have multiple semantically distinct separators, minimized formulae may not be logically equivalent to the non-minimized formula.

Formula	Not Minimized	Minimized
φ_1	$\exists v0 \exists v1.(f(v0) = v0 \vee f(v1) = v1)$	$\exists v0.f(v0) = v0$
φ_2	$\exists v0 \forall v1 \exists v2.(E(v2, v1) \wedge \neg(E(v0, v2) \wedge E(v1, v0)))$	$\forall v0 \exists v1.E(v1, v0)$
φ_3	$\forall v0 \forall v1 \exists v2.(E(v1, v0) \implies (E(v2, v0) \wedge E(v1, v2)))$	$\forall v0 \forall v1 \exists v2.(E(v1, v0) \implies (E(v2, v0) \wedge E(v1, v2)))$

Table 1: Examples of Minimized Formulas

5 Future Work and Conclusions

We have presented Folloy as a novel tool to solve the *constrained first-order separability* problem. It presents a more general approach compared to the existing method [3] allowing for guaranteed shortest formula, non-prenex formulas, and additional constraints.

The biggest limitation is the need for the environment encoding. Our encoding requires E^V hard-coded `Environments` where E is the maximum number of constants and V is the number of free variables. This means that Folloy is only suitable for examples with a relatively small number of constants. One solution we are currently exploring is to fix a prenex quantifier and use Alloy to generate the body of the formula. This approach reduces the runtime significantly as it doesn't require hard-coded environment and also opens up the possibility for parallelization with each Alloy instance. However, it would also sacrifice the generality as we are only restricted to prenex formulas.

As future work, we plan a small DSL for constraints (reducing Alloy-level errors) and a brief checklist for curating P/N (diverse boundary cases, near-miss negatives, and a minimal witness basis to cut solve time). Alloy$^{\text{Max}}$ also allows for other interesting features, such as constraint maximization. Instead of just positive and negative examples, we can add *semi-positive* or *semi-negative* examples where Alloy$^{\text{Max}}$ will try to satisfy as many as possible but does not need to satisfy all to output an instance. This allows for examples in which the user is unsure of its correctness and allows for extra expressiveness. Maoz et al. [5] similarly has an alternative option for synthesizing constructor and component models.

Acknowledgments

This work was supervised by Abigail Hammer, Ian Dardik, and Eunsuk Kang of Carnegie Mellon University. It was supported by National Science Foundation REU Program.

References

[1] Derek Egolf, William Schultz, and Stavros Tripakis. 2024. Efficient Synthesis of Symbolic Distributed Protocols by Sketching. arXiv:2405.07807 [cs.LO] https://arxiv.org/abs/2405.07807

[2] Mario Gleirscher and Diego Marmsoler. 2018. Formal Methods: Oversold? Underused? A Survey. *CoRR* abs/1812.08815 (2018). arXiv:1812.08815 http://arxiv.org/abs/1812.08815

[3] Jason R. Koenig, Oded Padon, Neil Immerman, and Alex Aiken. 2020. First-order quantified separators. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 703–717. https://doi.org/10.1145/3385412.3386018

[4] María Manzano and Víctor Aranda. 2022. Many-Sorted Logic. In *The Stanford Encyclopedia of Philosophy* (Winter 2022 ed.), Edward N. Zalta and Uri Nodelman (Eds.). Metaphysics Research Lab, Stanford University.

[5] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2014. Synthesis of Component and Connector Models from Crosscutting Structural Views. *CoRR* abs/1408.5696 (2014). arXiv:1408.5696 http://arxiv.org/abs/1408.5696

[6] Maria Spichkova. 2014. Human Factors of Formal Methods. *CoRR* abs/1404.7247 (2014). arXiv:1404.7247 http://arxiv.org/abs/1404.7247

[7] Changjian Zhang, Parv Kapoor, Ian Dardik, Leyi Cui, Romulo Meira-Goes, David Garlan, and Eunsuk Kang. 2024. Constrained LTL Specification Learning from Examples. arXiv:2412.02905 [cs.SE] https://arxiv.org/abs/2412.02905