TPLP: Page 1–8. © The Author(s), 2021. Published by Cambridge University Press 2021
doi:10.1017/xxxxx

Appendix of Submission 38 of ICLP 2025

ZIYI YANG, GEORGE PÎRLEA, and ILYA SERGEY
National University of Singapore

Appendix A One Tiny Example of Definition 1 in Sec. 2.1

This sections provides an example of the FO synthesis problem

Example 1 (*A tiny Example*)
Considering a FO signature $\Sigma = \langle C, R, F, S\rangle$ with C and F empty, $R = \{p, q, r\}$, and $S = \{X\}$. The synthesis problem is defined with the inputs:

- the set of FO formulas is restricted to the form of $\forall X : lit_1$ or $\forall X : lit_1 \vee lit_2$, which is a disjunction of one or two literals with one universal quantifier. Without the pruning, Ω_0 contains $(2 * 3) + (2 * 3)^2 = 42$ formulas.
- the set of FO structures is $\sigma = \{M_1, M_2\}$, where M_1 and M_2 are two models over Σ sharing the universe $\{x_0, x_1, x_2\} \in X$ and the interpretations of p, q, and r are:
  1. M_1: $[p^{M_1} = \{x_0,\ x_1\},\ q^{M_1} = \{x_1,\ x_2\},\ r^{M_1} = \emptyset]$
  2. M_2: $[p^{M_2} = \{x_0,\ x_1\},\ q^{M_2} = \{x_2\},\ r^{M_2} = \{x_1\}]$

And the output of the synthesis problem based on Definition 1 is a set of formulas

$$\Phi = \{\forall X.p(X) \vee q(X), \forall X.p(X) \vee \neg r(X)\}.$$

As a simple case of the sliced-template, our algorithm will first find satisfied formula in $\forall X : lit_1$ (and fails), then checking the satisfied formula in $\forall X : lit_1 \vee lit_2$ (with the two formulas successfully found). The inputs of FORCE, the bounded FO search space and the real traces of distributed system protocols, are much more complex. An interested reader can refer to `configs/` and `traces/` folders in https://github.com/verse-lab/FORCE.

Appendix B The Detailed Study on the Synthesis Taxonomy

In this section, we provide our detailed study on the synthesis techniques used in existing invariant inference tools outlined in Sec. 2.3.

B.1 System-level Aspects

Inference mode. This aspect determines the way the overall inference procedure uses FO formula synthesis. In particular, *one-shot* synthesis generates all satisfied formulas in the search space given fixed input examples (*e.g.*, sampled traces of a distributed protocol), while *multi-shot* synthesis generates a set of candidate formulas incrementally, based on examples obtained incrementally (*e.g.*, counter-examples of current invariants). Combined approaches use system traces or counter examples to guide the multi-shot synthesis, calling the synthesis procedure multiple times.

Historically, the majority of multi-shot synthesis algorithms can be seen as extensions of IC3 (Bradley 2011), while one-shot synthesis can be considered as extensions of Houdini (Flanagan and Leino 2001). Crucially, both modes share the underlying FO synthesis problem similar to the definition of our synthesis problem, which means our FORCE framework can be used to improve most existing approaches.

Language restriction. To make the synthesis problem tractable, it is frequently defined for a subclass of first-order logic. The most common one is the Effectively Propositional Logic (EPR) fragment, a subset of FOL in which formulas can be transformed into equivalent propositional formulas, allowing for provably decidable verification.

While EPR provides a theoretical decidability guarantee, in practice, synthesis approaches often impose additional other syntactic constraints. For example, the k-pseudo-DNF proposed in P-FOL-IC3 (Koenig et al. 2022) is a syntax restriction for practical efficiency: it is based on the observation that the invariant formulas written in such form are smaller than standard DNF, reducing the search space that needs to be explored. Specifically, a k-pseudo-DNF formula has the matrix in the form of $c_1 \rightarrow (c_2 \vee \ldots \vee c_k)$, where c_i is a conjunction of literals. This is essentially a heuristic of the search, which makes sense because implications are commonly used to express invariants.

DistAI (Yao et al. 2021) proposed sub-templates for efficiency, exploiting the following property of first-order logic with equality:

$$\forall X_1, X_2 : T.mat(X_1, X_2) \equiv \forall X : T.mat(X, X) \wedge \forall X_1 \neq X_2 : T.mat(X_1, X_2) \quad (0)$$

With such a property, formula synthesis can avoid enumerating formulas on the LHS of Eq. 0 because they are equivalent to the conjunction of the two RHS formulas. Therefore, the only enumerations among two sub-templates on the right are required to synthesis the satisfied formula on the left, resulting in an accelerated enumeration.

However, upon close examination of existing approaches' implementations, we discovered that such syntactic constraints are not extensible in many existing invariant inference tools. For example, extending k-pseudo-DNF with sub-templates would require one to modify any algorithm that manipulates formulas in P-FOL-IC3.

B.2 Algorithm-level Aspects

Since the search space of first-order formulas is large and thus intractable for a brute-force search, existing tools introduce algorithmic pruning techniques. We characterise pruning techniques according to the remaining three aspects in Sec. 2.3. Since the last one have been detailed in Sec. 6, Here we discuss the first two aspects.

Redundancy elimination. The most common pruning to apply is redundancy elimination, which is often used in synthesis. The idea is to simply eliminate the formulas that are equivalent to each other. For many tools, approximations such as "pruning by symmetry" (shown in Sec. 3) are used, which uses the symmetry of formulas under quantification to identify and prune away redundant ones. More than the syntactic-based equivalence, the redundancy can also be introduced by semantics (*e.g.*, tautology and contradiction), where many case-by-case rules are used in DuoAI. An example of formulas eliminated by this kind of redundancy is the one containing $p(X) \wedge \neg p(X)$ or $p(X) \vee \neg p(X)$ as sub-

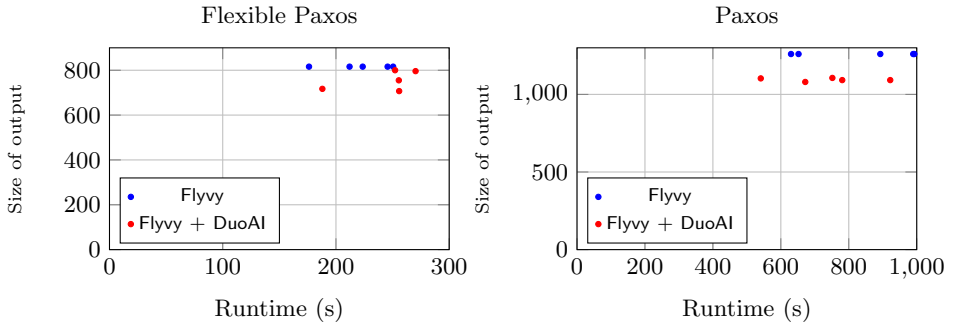

Fig. C 1: Optimising Flyvy via DuoAI's output.

expressions. Another approach to redundancy is elimination *canonicalisation*, which is implemented by Flyvy (Frenkel et al. 2024): it also uses symmetry breaking, but defines a partial order of formula sets (instead of the individual formulas) to eliminate redundancy.

A more complex but very effective pruning strategy of this category is proposed in DuoAI (Yao et al. 2022), where a large number of DNF formulas are shown to be redundant by their *decomposition*: $A \equiv B \wedge C$. Similar to Eq. 0, the formula A can be decomposed into a conjunction of smaller formulas, so the original formula can be pruned if all the smaller formulas are in the search space. That said, the authors of Flyvy (*cf.* Appendix D of Frenkel et al. 2024) found that the decomposition in DuoAI is unsound in certain cases (*i.e.*, it leads to over-pruning), and proposed an amended version. Unfortunately, we found it not easy to switch the DuoAI implementation to the amended version, as it uses the intermediate results of the original decomposition in the overall synthesis loop. This also makes it challenging to fairly compare the efficiency of the two methods.

Incremental pruning. Another inherent part of nearly all efficient approaches to FOL formula synthesis is incremental pruning. The idea is to test formulas against the set of input FO structures in a specific order, exploiting the entailment relation between formulas to eliminate the need to test some of them.

As introduced in Sec. 4.1, the *implication graph* is an effective technique of this aspect. More than this, DuoAI also identify another incremental pruning technique is based on *co-implication*, which combines the intermediate results of FO model checking with the redundancy elimination to prune the search space. For example, if we find that formula $\forall x.P(x) \Rightarrow R(x)$ satisfies all examples, we do not need to the test any formulas of the form *prefix*. $(P(x) \wedge R(x) \wedge F) \vee G$, since they are co-implied by the original formula and *prefix*. $(P(x) \wedge F) \vee G$.

The two kinds of pruning discussed above are essential in accelerating the search of FO formulas. Their correctness is justified by the properties of FOL, such as entailment and equivalence. Importantly, this means that one can see the synthesis of FO formulas as enumeration *modulo* property-based pruning.

Appendix C The Additional Statistics of Sec. 5.3

As said in Sec. 5.3, there are only two non-trivial protocols in Flyvy's benchmark that have quantifier alternation, for which we can produce comparable results. For both protocols, we execute the original Flyvy and the optimised Flyvy for 5 times with the sizes of the

output (been discussed) and the runtime shown in Fig. C 1. Performance-wise, we did not achieve notable improvement on the runtime of Flyvy, which is not unexpected, since the bottleneck of Flyvy is not the synthesis; instead, the SMT solving in Flyvy dominates the runtime and makes the runtime unstable.

References

Bradley, A. R. SAT-Based Model Checking without Unrolling. In *VMCAI* 2011, volume 6538 of *LNCS*, pp. 70–87. Springer.

Flanagan, C. and Leino, K. R. M. Houdini, an Annotation Assistant for ESC/Java. In *FME* 2001, volume 2021 of *LNCS*, pp. 500–517. Springer.

Frenkel, E., Chajed, T., Padon, O., and Shoham, S. 2024. Efficient implementation of an abstract domain of quantified first-order formulas. *CoRR*,, abs/2405.10308.

Hance, T., Heule, M., Martins, R., and Parno, B. Finding Invariants of Distributed Systems: It's a Small (Enough) World After All. In *NSDI* 2021, pp. 115–131. USENIX Association.

Koenig, J. R., Padon, O., Shoham, S., and Aiken, A. Inferring invariants with quantifier alternations: Taming the search space explosion. In *TACAS* 2022, volume 13243 of *LNCS*, pp. 338–356. Springer.

Yao, J., Tao, R., Gu, R., and Nieh, J. DuoAI: Fast, Automated Inference of Inductive Invariants for Verifying Distributed Protocols. In *OSDI* 2022, pp. 485–501. USENIX Association.

Yao, J., Tao, R., Gu, R., Nieh, J., Jana, S., and Ryan, G. DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols. In *OSDI* 2021, pp. 405–421. USENIX Association.